TU WIEN Informatics

Visualization-Guided Classification of Carbonized Seeds from Early Human Civilizations

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Andreas Gogel, B.Sc.
Matrikelnummer 0801243

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Prof. Eduard Gröller
Mitwirkung: Dr. Manuela Waldner

Wien, 25. November 2020

Andreas Gogel

Eduard Gröller

TU WIEN Informatics

Visualization-Guided Classification of Carbonized Seeds from Early Human Civilizations

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

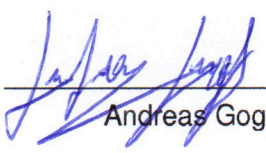
Andreas Gogel, B.Sc.
Registration Number 0801243

to the Faculty of Informatics

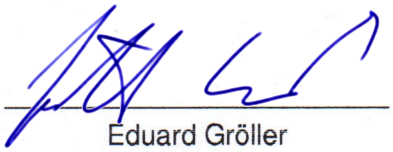
at the TU Wien

Advisor: Prof. Eduard Gröller
Assistance: Dr. Manuela Waldner

Vienna, 25th November, 2020



Andreas Gogel



Eduard Gröller

Erklärung zur Verfassung der Arbeit

Andreas Gogel, B.Sc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 25. November 2020

Andreas Gogel

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich bei dieser Arbeit unterstützt haben. Zuerst möchte ich mich bei meinem Betreuer Prof. Eduard Gröller und meiner Betreuerin Dr. Manuela Waldner für ihre kostruktive Hilfe bedanken. Außerdem möchte ich mich bei Dr. Yannick Städler bedanken, der sowohl die Daten bereitgestellt, als auch sein biologisches Wissen mit mir geteilt hat. Zuletzt gilt es noch meiner Familie und Freunden einen Dank auszusprechen, die mir immer Rückhalt gegeben haben.

Acknowledgements

At this position, I want to thank all who supported me during this work. At first, I thank my supervisors Prof. Eduard Gröller and Dr. Manuela Waldner for their constructive help. Secondly, I thank Dr. Yannick Städler for the data and his biological knowledge he shared with me. In the end, I thank my family and my friends who always back me up.

Kurzfassung

Seit der Neolithischen Revolution, vor ca. 10.000 Jahren, sind Nutzpflanzen ein wichtiger Bestandteil unserer Ernährung. Wissenschaftler*innen der Archeobotanik versuchen Pflanzensamen zu finden und zu bestimmen, die bereits von der Menschheit in der Vergangenheit genutzt wurden. Viele der gesammelten Proben sind durch einen Verkohlungsprozess erhalten geblieben, wodurch aber ihre Form und innere Struktur verändert wurden. Der Grad der Veränderung ist bestimmt durch die Temperatur und die Heizdauer. Im normalen Ablauf wird ein*e Experte*in hinzugezogen welche*r die Pflanzensamen bestimmen kann. Da es relativ wenige von diesen Experten gibt, liegt der Wunsch nach einer Automatisierung nahe. Das Ergebnis dieser Arbeit ist eine Software, die die Computed-Tomography-Scans (CTs) von Pflanzensamen laden, die Pflanzensamen in den Dateien automatisch segmentieren und separieren, verschiedene Merkmale als Deskriptoren daraus extrahieren und damit einen Klassifikator trainieren kann. Um die vom Klassifikator erhaltenen Wahrscheinlichkeiten überprüfen zu können, sind Visualisierungen verfügbar, die es ermöglichen den Einfluss der extrahierten Merkmale auf die Klassifizierung nachzuvollziehen. Damit die Pflanzensamen auch direkt betrachtet werden können und um einen guten Überblick zu erhalten wie Medianobjekte von jeder Klasse aussehen, können diese mit Hilfe von verschiedenen Volumen-Visualisierungstechniken angezeigt werden. Die Ergebnisse eines Kreuzvalidierungsverfahrens mit 1043 bekannten Elementen zeigen, dass der Klassifikator eine Genauigkeit von 85 Prozent erreicht. Die während des Lernprozesses falsch klassifizierten Datenpunkte werden zusätzlich dargestellt, um es den Experten zu ermöglichen, zu erkennen, welche Klassen besonders inakkurat sind und wo sich diese Datenpunkte bezüglich der Features befinden. Als besonders hilfreich für den/die Domain Experten*innen hat sich die Möglichkeit herausgestellt, die extrahierten Merkmale in eine Comma Separated Values (CSV) Datei zu exportieren und die Ausgabe der Klassifizierungen als Wahrscheinlichkeiten pro Spezies anzeigen zu lassen.

Abstract

Since the Neolithic Revolution approximately 10.000 years ago, crop plants are an important part of our food. Researchers of archeobotany try to find and determine the species that humankind used already in the past. Most of the gathered samples are preserved due to carbonization, but the shape and inner structures are deformed because of this process. The amount of distortion is given by the temperature and the time they are heated. Normally, an expert is consulted to classify them. Since there are only a few experts in this field, an automatic approach is requested. The result of this work is a software, which can load the Computed Tomography (CT) scans, segment and separate the seeds within the samples, calculate different shape features as descriptors, and train a classifier. To have an overview of how the seeds look like, different volume visualizations are available to show selected samples or median seeds of each class. To validate the probabilities of the learner, additional visualizations are available, which show the influence of the extracted features on the classification. A cross validation method with 1043 known samples results in a classification accuracy of 85 %. The incorrectly classified samples of the ground truth are visualized to display the expert user where they are located regard to the extracted features and which results are especially inaccurate. It turned out, that the opportunity to export the features into a tabular filetype and the visualization of the output probabilities of the classifier for each species were particularly helpful for the domain experts.

Contents

CHAPTER 1

Introduction

Botanical archeology (Archeobotany) is a field of archeology specializing on the origin of agriculture, domestication, and other human-plant interactions in the past [VdV18]. One research aspect of botanical archeology lies in the investigation of ancient seeds. From archaeological discoveries of single seeds, biologists try to analyze the evolution of seeds and thereby derive plausible explanations for the domestication of plants. Another important question to them is, which species were used first and how they changed over time. All these are nontrivial tasks because of the condition of the gathered seed samples. To survive until today, there are only a few options. One of them is carbonization. In this condition, seeds will not pullulate, get eaten or rot, because the organic substances are all removed thermally.

The task of this work is to support the classification process and create descriptions of gathered ancient seeds, because the main goal of the domain expert is to create a big database of these seeds for further investigations. Due to the seeds' condition, it has to be assumed that the machine learner will make several miss classifications. For this reason, visualizations would be offered to check the suggestions. The data consist of 3D volumes with mostly multiple seeds per file, which have to be separated. There are also partial fragments of seeds or other artifacts that have to be considered. After the segmentation and separation, several shape features are calculated, additionally with distribution information and shown in additional visualizations. In the end, a machine learner is trained to offer probability values of all classes for new samples.

In the following list, the main challenges are given.

- Loading the data: Both of the input data, labeled and unlabeled samples, containing different datatypes which should be possible to load. Additionally, there will be far more than 1000 samples from more than 100 species in the ground truth, with a

filesize of up to 22 GB per file. Concerning the storage condition of the physical seeds, there is more than one seed per file which has to be segmented and separated to calculate the features for each of them.

- Feature extraction: Since the seeds are carbonized by different unknown heats, the amount of surface destruction differs as well. Features have to be found which are most discriminative for a good classification and for storage purposes. Because the seeds in the scans are not oriented equally, only rotation invariant features can be used.

- Machine learning: There are classes that contain over 200 samples and others with only one sample. A learner technique has to be determined, which can do multiclass classification and can handle a highly unbalanced training dataset. The machine learning part is included to offer the domain expert a list of probabilities. This way, an unknown seed can be assigned to the class it belongs to as fast as possible. Since the samples are distributed very unbalanced over the classes and the seeds are very distorted regarding to carbonization process, the assumption is that the accuracy would not be good. Because of this the next point is added to the requirements.

- Visualizations: Visualizations: For a better overview of all classes, to check the descriptor values and to examine the suggestions of the machine learning part, additional visualizations should be available. Initially, the seeds should be viewable as they are in the data. For this, different Direct Volume Rendering (DVR) techniques are considered. To have an idea how a seed of a class would look like, the median seeds with respect to two user selected features should be shown. To find clusters in the data, a scatter plot has to be shown. This scatter plot should also contain the convex hulls for every class to get a better overview of the location of each class within the plot and to detect outliers easily. It is also interesting for the domain expert to know, how particular features of the seeds are distributed. The Kernel Density Estimation (KDE) and a binned histogram can be considered to visualize these distributions in the background of the scatter plot.

- Data saving: It should also be possible to store all information created for later use. Otherwise, every step would have to be repeated after every restart of the software. It is important in this requirement, that the segmented and separated seeds use much less space than the original files.

In summary, the domain expert should be guided during the classification process of an unknown scanned seed into different species (or at least genus) and should get descriptors of the shape of the seeds as additional information in the seed database.

In this work, initially the biological backgrounds are given in Chapter 2. In Chapter 3, methods are described, which are used or closed related of the software. The state of the art methods concerning to this work are discussed in Chapter 4. The main part of this work is Chapter 5 that contains the details about the components of the software and

their pipelines. Since visualizations are used in different parts of the software, they are described in the associated sections. Thus the methods are clearly defined previously in Chapter 3, the used techniques are only defined shortly and referenced in the appropriate sections. The libraries used and other implementation details are described in Chapter 6. Finally, in the Chapter 7 and Chapter 8, the results and the conclusion are discussed. The conclusion contains also tasks for future work.

CHAPTER 2

Archeological Background

Approximately about 10.000 years ago, humans began the transition from a hunter and gatherer to a settlement society [Wei05]. This process is called the Neolithic Revolution and one of the hot spots of this shifting lies in the Fertile Crescent (in Figure 2.1 a map of this area is shown)[BJPA09]. The Fertile Crescent is an area located in the Near East (former Mesopotamia). The rivers Euphrates and Tigris have a big impact on this area and transport nutrients their soil. Back then the climate was also advantageous for the plants.

The ancient people found out that they do not have to search eatable plants for themselves and their domesticated animals, if they can grow these plants around their houses. Animal domestication took place around this time as well. This way, they could control the amount of food they have and the problem of finding edibles was minimized. Over time, they recognized that bigger plants come from seeds of big individuals and they started out to select specific plants to breed better species. This was the beginning of agriculture, the human race started to expand. In the area of the Fertilize Crescent, mainly species of *Vicia* (vetch), *Lens*, *Lathyrus* (vetchlings), or *Pisum* (peas) are used, because they are a good source of proteins and carbohydrates.

Researchers in the field of Botanical Archeology are trying to understand which plant species were selected and how they changed gradually. New species are discovered as eatable because of human selection. Most of the remaining seeds from these very early times are carbonized. This way, they could be preserved. However, in this condition, it is hard to identify the species of the seeds. The reasons for the carbonization are often rituals [AB11], inexperienced cooking, or uncontrolled fire e.g., house fire) [CFWJ15]. During the burning process, the seeds are deformed and loose surface details. An example scheme of the procedure is shown in Figure 2.2. It is easy to see that this condition makes it challenging to classify the original species from these samples. Another interesting aspect for the researchers is that some of the ancient species are already extinct. In this

Figure 2.1: A map of the Fertile Crescent [BJPA09], the origin of the ancient seeds used in this work.

case, a classification of the species is not possible, and the experts are trying to find out the botanical genus, so seeds of this species can be put in an evolutionary context with other seeds. The botanic genus is a step in the ordering of biological organisms called taxonomy and systematics [Sim19]. Organisms with equivalent properties are grouped together, and relationships between species are more visible in this naming system. The main categories of this system can be seen in Figure 2.3. For a better understanding of the carbonization process, researchers try to reproduce it under lab conditions [MR08]. Seeds from different species act differently during the heating process. Regarding their importance for humans, these processes are well investigated for food grains. These synthetic processes were also used to get a robust ground truth for this work. The botanist laid the seeds in a ceramic crucible and heated them for a specific time in a muffle furnace. To get different conditions, some of the crucibles are covered in aluminum foil. For our purpose, the resulting seeds are 3D scanned with Computed Tomography (CT) by a domain expert. To store the physical seeds properly, they are pooled together into polymers in groups of up to six seeds of the same species. These polymers are used because they have good background properties for the CT scanning.

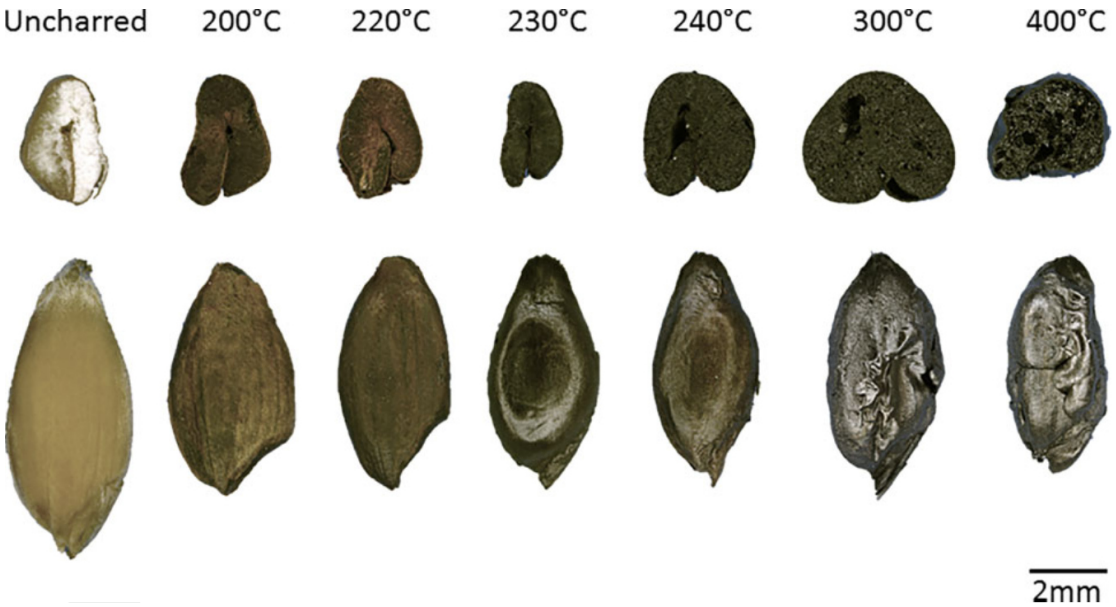


Figure 2.2: A scheme of the carbonization process using the example of an einkorn seed heated between 200° and 400° Celsius [CFWJ15]. The top row represents a cross-section in top-view and the bottom row the seed in side-view.

This creates another challenge because, for the later feature calculation, the individual seeds have to be segmented and separated virtually.

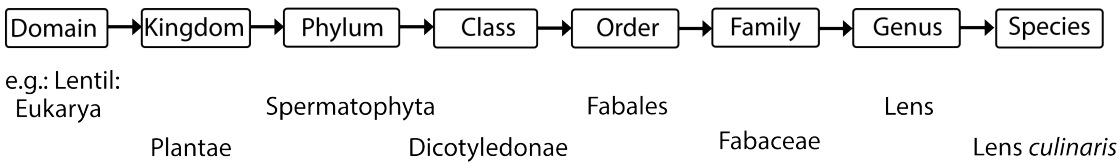


Figure 2.3: Scheme of the main taxonomy groups in biology (sub-groups not included). An example of the Lentil species *Lens culinaris* can also be found (taxonomy information is taken from the Cabi database [Cab].

CHAPTER 3

Computed Tomography Data Analysis

In this chapter the technical background of different analysis methods for 3D data is discussed. Initially, the acquisition of the data with a CT scanner is described (Section 3.1) and the properties of the used data are discussed (Section 3.2). The processing steps are explained in Section 3.3. For the analytical part, it is important to calculate numerical features from the data samples, so they can be used in the machine learner part as well as in the visualizations. Different features are introduced in Section 3.4. Handling data with a lot of features, it is common to reduce the dimensionality of them to get a better overview and check whether the features separate the different classes well or not. For this purpose, the Principle Component Analysis (PCA) will be illustrated (Section 3.3.3). At the end of this chapter, classification techniques for machine learning are discussed (Section 3.7). In the user interface, which is an important part of this work, the data samples are visualized as they are, therefor a section about volume rendering has been included (Section 3.8).

3.1 Acquisition

The technology of scanning is based on X-Rays, discovered in 1895 by Wilhelm Conrad Röntgen [CM11]. Rays are cast from different angles through the target objects and are measured with sensors on the opposite side. Some structures of the target objects absorb more radiation than others. By measuring the radiation on the detector, it is possible to get an image. Since applying this procedure once will only result in one slice (i.e., a single image) of the object, it is necessary to move the scanner or the object. The result is a 3D dataset (volume) from the inner structures of the object. The volume contains density values. After normalization of the density values, one means full absorption because of very dense materials and zero stands for no absorption. Contrast agents, which absorb

more radiation, may also be used to highlight specific areas. A scheme of the basic mode of functioning of an industrial CT scanner is shown in Figure 3.1.

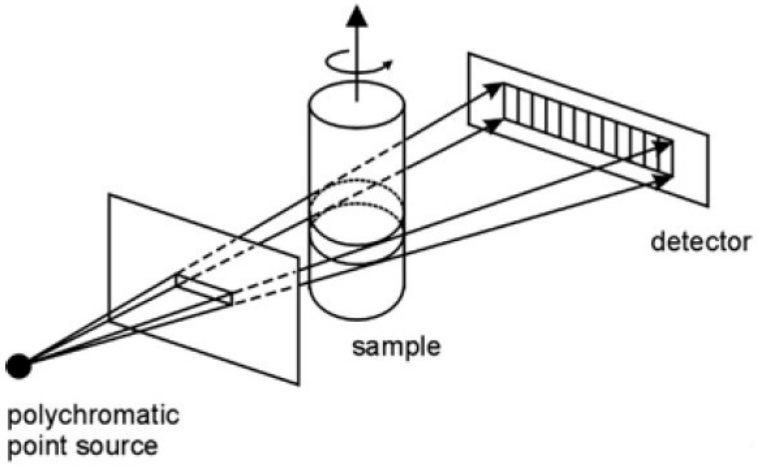


Figure 3.1: Principal functionality of an industrial CT-scanner [CM11]. X-Rays are shot slice by slice from the source through the target object. The sensors on the other side measure the amount of unabsorbed radiation. To get a full 3D volume of the object, the target object is rotated.

3.2 Data Properties

The data consists of files with single or multiple seeds. The resolution of single-seed files is approximately $180 \times 180 \times 180$ voxels ($width \times height \times depth$) which requires about 16 MB space. These values depend strongly on the size of the scanned seed. In the test dataset of this work, all of the files are in the Tagged Image File Format (TIFF) data type, a file-type where it is possible to pack more than one image into one file [Ass92]. The multiple-seed files are volumes of size $400 \times 400 \times 400$ voxels in DICOM format (up to 22 GB space per file). Digital Imaging and Communications in Medicine (DICOM) is a standard format for medical images and has the benefit of combining different information about the containing data [MDG08]. Since CT equipment is often used for medical purpose, the DICOM format is implemented in nearly all of them. The sample per class variation generates also an unbalanced classification problem. Table 3.1 shows the used data.

3.3 Processing

In this section, methods are described, which are essentially preprocessing steps for the feature extraction. The order of them follows their appearance in the processing pipeline described in Chapter 5. Initially, different filtering methods are described to

Genus	Species	Count	State	Type
Lens	*culinaris*	109	fresh	TIFF
	culinaris	85	carbonized	TIFF
	culinaris	8	carbonized	DICOM
	ervoides	35	carbonized	DICOM
	nigricans	29	carbonized	DICOM
	orientalis	33	carbonized	DICOM
Pisum	*sativum*	289	fresh	TIFF
	sativum	138	carbonized	TIFF
Vicia	*sativa*	36	fresh	TIFF
	sativa	6	carbonized	DICOM
Lathyrus	*clymenum*	42	fresh	TIFF
	alphaca	28	carbonized	DICOM
	annuus	7	carbonized	DICOM
	articulatus	5	carbonized	DICOM
	aureus	1	carbonized	DICOM
	boissieri	9	carbonized	DICOM
	cassius	3	carbonized	DICOM
	cicera	9	carbonized	DICOM
	clymenum	12	carbonized	DICOM
	czecyottianus	3	carbonized	DICOM
	digitatus	6	carbonized	DICOM
	grandiflorus	3	carbonized	DICOM
	hirsutus	13	carbonized	DICOM
	inconspicuus	14	carbonized	DICOM
	incurvus	12	carbonized	DICOM
	nissolia	9	carbonized	DICOM
	pallescens	5	carbonized	DICOM
	palustris	3	carbonized	DICOM
	pratensis	4	carbonized	DICOM
	roseus	4	carbonized	DICOM
	saxatilis	4	carbonized	DICOM
	setifolius	8	carbonized	DICOM
	spathulatus	20	carbonized	DICOM
	sylvestris	8	carbonized	DICOM
	tuberosus	8	carbonized	DICOM
	vernuus	4	carbonized	DICOM
	vinealis	16	carbonized	DICOM

Table 3.1: Used data with numbers of samples, status and filetype.

prepare the data for firther steps, followed by the segmentation and separation approaches. An approximate registration method to get the seeds to an equal rotation will also be described. The registration is only for the visual purpose, because the features should be rotation invariant.

3.3.1 Otsu Threshold

In the later (Section 3.3.2) described Connected Component Analysis (CCA) to segment the single seeds, it is important to have a binary image that only contains zero and ones. This section describes how to calculate an automatic threshold using the Otsu method [Ots79]. It is used for image or volume binarization or filtering to avoid a static threshold. The calculation uses an analysis technique on the histogram of the data. The Otsu method finds an optimal threshold by separating the data into two classes by minimizing the inner class variance. Initially, a histogram of the data is needed. A histogram (**hist**) is a vector whose elements are the number of values within the image/volume which are (nearly) equal. For example, for a gray value image with values in $[0, 255]$, a histogram vector with 256 elements can be created. The fifth element in this vector contains the number of pixels with the gray value of 4. Is the range bigger, it is also possible to summarize values of a specific range into one bin. An element contains then the number of values between a certain range. An index into the histogram vector can be calculated with: $index = \frac{value}{maxValue} \cdot (binCount - 1)$ if the index starts with zero. Since only the histogram is used, spatial information of the original image data is lost. Two pixels with the same value are treated equally, although they have different local meanings in the image (e.g., noise, background or foreground). In the algorithm, the histogram is partitioned into two classes with an index thres. The weighted variance is used to measure the quality of the split and the index that fits best is the outcome threshold. Equation 3.1 shows the formula for the weighted variance respective to the index *thres* of the histogram:

$$\sigma_w^2(thres) = q_1(thres) \cdot \sigma_1^2(thres) + q_2(thres) \cdot \sigma_2^2(thres), \tag{3.1}$$

where q_1, q_2 are the probability values that a random point would fall into the first or the second partition and σ_1^2, σ_2^2 are the inner class variances of the two classes. The probability values can be calculated by the use of Equation 3.2:

$$q_1(thres) = \frac{1}{N} \cdot \sum_{i=0}^{thres} hist(i) \quad , \quad q_2(thres) = \frac{1}{N} \cdot \sum_{i=thres+1}^{binCount-1} hist(i). \tag{3.2}$$

N is the total number of values in the histogram to normalize the sum of all elements to one and $hist(i)$ is the value of the histogram on position i. In the next step, the mean values of the two classes are needed to be calculated, as shown in Equation 3.3:

$$\mu_1(thres) = \frac{1}{N_1} \sum_{i=0}^{thres} i \cdot hist(i) \quad , \quad \mu_2(thres) = \frac{1}{N_2} \sum_{i=thres+1}^{binCount-1} i \cdot hist(i). \tag{3.3}$$

The variables N_1 and N_2 represent the number of elements of the respective class. Now all parameters are given to calculate the inner class variances. The calculation is given in Equation 3.4:

$$\begin{aligned}\sigma_1^2(thres) &= \frac{1}{N_1}\sum_{i=0}^{thres}(i-\mu_1(thres))^2\cdot hist(i)\\ \sigma_2^2(thres) &= \frac{1}{N_2}\sum_{i=thres+1}^{binCount-1}(i-\mu_2(thres))^2\cdot hist(i)\end{aligned} \tag{3.4}$$

The Otsu algorithm iterates through all elements of the histogram and determines where $\sigma_w^2(thres)$ is minimal. The $thres$ parameter holds then the best threshold. A faster approach, because of lesser calculation steps, which results in the same outcome uses the between-class variance σ_b^2 [BN16]. Starting with the formula of the overall variance ($\sigma^2 = \sigma_w^2(thres) + \sigma_b^2(thres)$), σ_b^2 can be calculated with Equation 3.5:

$$\sigma_b^2(thres) = q_1 \cdot q_2 \cdot (\mu_1(thres) - \mu_2(thres))^2. \tag{3.5}$$

q_1, q_2, $\mu_1(thres)$ and $\mu_2(thres)$ can be calculated as already shown in Equation 3.2 and Equation refeq:meanOtsu. The threshold $thres$ which maximize the between-variance $\sigma_b^2(thres)$ is then the optimum. Figure 3.2 shows outputs of this algorithm on example seed volume data.

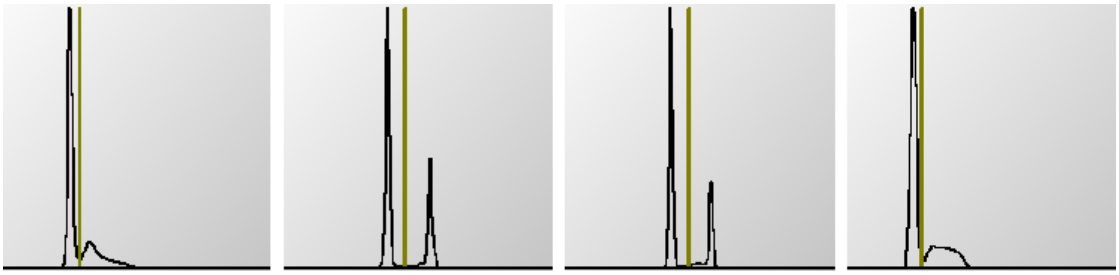


Figure 3.2: Otsu threshold (dark yellow) calculations of the sample seed volumes. The left peak contains the backgroundnoise and the right peak the actual data information which are the seed densities.

3.3.2 Filtering

Since the 3D volume data contains noise and has a high resolution, some filtering methods can be used to optimize and simplify the data. Some tasks, like the Connected Component Analysis (CCA), do not need the exact data values and are easier to solve if the input is a binary image. So a binarization filter can be used. These steps can improve and speed up the results of the later methods like the seed segmentation and noise filtering. A lot of the filtering methods use a filter kernel. Let $\mathbf{D}$ be the 3D volume data matrix with dimension $n \times m \times l$. Let also $\mathbf{C}$ be a $b \times b \times b$ filter kernel matrix where b an

odd number greater than zero. The filter kernel is shifted over every element of the data matrix $\mathbf{D}$ in a way that $\mathbf{C}(\lfloor\frac{b}{2}\rfloor, \lfloor\frac{b}{2}\rfloor, \lfloor\frac{b}{2}\rfloor)$ covers successively every element $\mathbf{C}(i, j, k)$ ($i \in [0, n-]$, $j \in [0, m-1]$ and $k \in [0, l-1]$). This operation is called convolution. To not interfere future calculations, a new output matrix $\mathbf{D}'$ is used to store the new values calculated from the covered elements within the kernel. This calculation differs from method to method. A problem that occurs are the elements on the edges. If, for example, $\mathbf{C}(\lfloor\frac{b}{2}\rfloor, \lfloor\frac{b}{2}\rfloor, \lfloor\frac{b}{2}\rfloor)$ covers the element $\mathbf{D}(0, 0, 0)$, some of the entries of $\mathbf{C}$ lie outside of the dimensions of $\mathbf{D}$. There are multiple options to handle those cases. The first one slides the kernel only in the range $([\lfloor\frac{b}{2}\rfloor, n - \lfloor\frac{b}{2}\rfloor], [\lfloor\frac{b}{2}\rfloor, m - \lfloor\frac{b}{2}\rfloor], [\lfloor\frac{b}{2}\rfloor, l - \lfloor\frac{b}{2}\rfloor])$. The values are all correct, but the result matrix will shrink by $b - 1$ elements in each dimension. Other methods would fill the entries outside the bounds with zeros or mirror the elements from inside the data matrix.

Median Filtering

The median filter is a method to handle the so-called salt-and-pepper noise [ZGWP16]. This noise specifies a pixel pattern which contains randomly high or low values within the data. The median value is defined as the middle value of the data. Respectively half of all elements in the data are lower or equal and half are greater than or equal to the median. As described above, this method uses a filter kernel $\mathbf{C}$. The new values $\mathbf{D}'(i, j, k)$ are calculated by sorting all values of $\mathbf{C}$ and take the $\lfloor\frac{d^3}{2}\rfloor$th element of this sorted list. Outliers are automatically filtered out. For the border point difficulty, it is possible to ignore all points outside of $\mathbf{D}$ and change the number of elements in these cases. A positive effect of this filter is, that the resultvalues are all part of the original data. On the negative side, fine structures, like thin edges, are also erased. There are some improvements to this algorithm like using weights on each element of the kernel and using edge-preserving techniques [ZGWP16]. Figure 3.3 shows an example filtering of a seed volume.

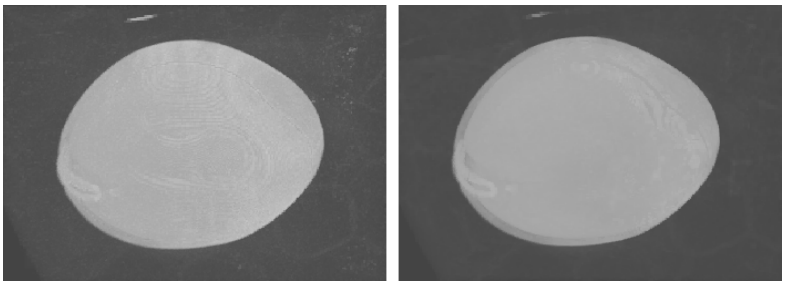


Figure 3.3: This image shows an original (left) and median filtered (right) seed volume. In median filtering the volume is convoluted with a filter kernel and the median of the elements within the filter is taken as the result value.

Average Filtering

With average filtering, the new value is calculated as the average of all elements (the sum of all values divided by the number of values) during the convolution process. The same effect, but computational more inefficient because of additional divisions, can be reached if the elements within the filter kernel are weighted by $\frac{1}{d^3}$ (with d being the size of the filter kernel) and simply summed up. Thereby, the amount of intensity variation is reduced and the noise within the data is smoothed. On the other hand, edges, textures, and other structures of interest get blurred. To avoid this phenomenon an edge-sensitive approach can be used. For this, the elements in the kernel are divided into eight directions. At first, the mean and the variance value for each direction is calculated. The resulting value is the average of the direction with the smallest variance. Because the directions with edges will have a higher variance, averaging across edges is avoided. An example filtering is shown in Figure 3.4.

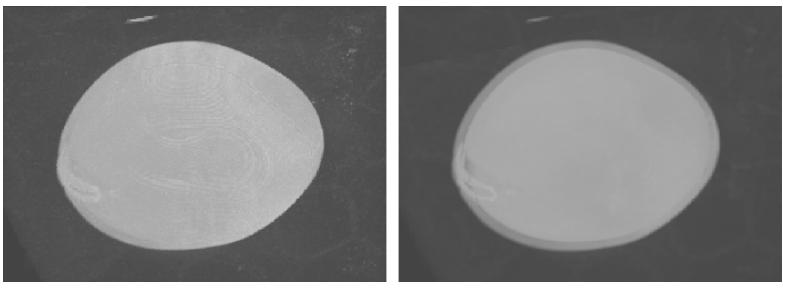


Figure 3.4: The image shows an original (left) and average filtered (right) seed volume. In average filtering the volume is convoluted with a filter kernel and the average of the elements within the filter is taken as the result value.

Gauss Filtering

It is also possible to sum up the elements weighed by a Gaussian distribution. The best way is to split the filtering into three 1D operations (one in each direction) with a 1D filter kernel. The formula is given in Equation 3.6, where x is the element of the 1D filter kernel and σ is the standard deviation of the distribution and can be user defined:

$$\frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma} \cdot e^{-\frac{x^2}{2 \cdot \sigma^2}}. \tag{3.6}$$

This way, the number of arithmetic operations is minimized. A technique, that uses the Gaussian distribution as weights and is edge-preserving is the bilateral filter [PIV16]. For an example of the Gauss and the bilateral filter see Figure 3.5.

Binarization

A binary 3D data matrix **Bin** is defined as a matrix whose elements $\mathbf{Bin}(i, j, k)$ are either zero or one [BN16], whereby the zero elements are tagged as background and

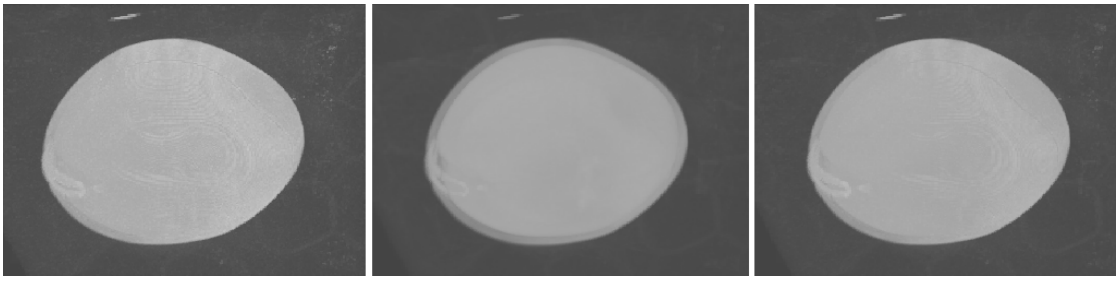


Figure 3.5: The image shows an original (left), a Gaussian filtered (middle) and bilaterally filtered (right) seed volume. In Gaussian filtering the volume is convoluted with a filter kernel and the weighted sum of the elements within the filter is taken as the result value. The weights are Gaussian distributed which smooths the volume during the filtering process. In the bilateral case, values are not smoothed over high gradients which preserves edges.

the foreground/interest elements are the items with the value one. A threshold is used to determine the class where the entries of **Bin** belong. The threshold can be an expert/user-chosen fixed value or calculated from the data. The already introduced Otsu threshold (Section 3.3.1) is a possible solution for the estimation. The binarization algorithm iterates through all indices of the data matrix $\mathbf{D}$ and the new value is zero if $\mathbf{D}(i, j, k) < threshold$ and otherwise one. A sample binarization is shown in Figure 3.6.

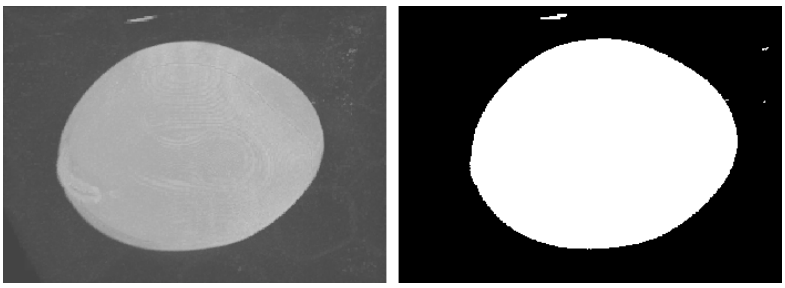


Figure 3.6: This image shows an original (left) and a binary filtered (right) seed volume. To get a good threshold for the binarization process, the Otsu algorithm is used.

Connected Component Analysis

To segment the single seeds within the files, a method is needed to assign equal labels to pixels/voxels that belong to the same object. The target of Connected Component Analysis (CCA) or Connected Component Labeling (CCL) is the generation of a data structure in which every connected set of pixels is assigned to a unique label [Sha96]. The input of techniques in this field is a binary data matrix (some of them can handle multiple entries per pixel as well). In 3D, there are two definitions of connection of two voxels. Firstly, the 6-neighborhood (similar to the 4-neighborhood in 2D) where only

the direct adjacent voxels $(i-1, i+1, j-1, j+1, k-1, k+1)$ are taken into account. Secondly, the 26-neighborhood (8-neighborhood in 2D), where also the diagonal elements are included. A scheme of the two different neighborhoods is shown in Figure 3.7.

(a) 6-Neighborhood (b) 26-Neighborhood

Figure 3.7: A scheme for the 6- and the 26-neighborhood of a voxel/element of a 3D matrix.

The first technique to determine the different labels calculates one label after another [AQIS07]. The data is scanned element by element. If a foreground pixel is found and it is not labeled yet, it gets a new label and is put into a queue. As long as there are items in the queue, the next step is performing. In this step, the neighborhood of the first element of the queue is checked. Neighboring voxels (pixels in 2D) that are background or already labeled are ignored. The other voxels/pixels are marked with the same label as the starting pixel and put into the queue. The first element is removed and the next one is chosen. If the queue is empty, the algorithm continues with step one and searches for the next unlabeled non-background pixel, but with an increased label number. This way, the data is only scanned once and multiple iterations are not necessary. The procedure is also described in Algorithm 3.1.

Another method that scans the data only once uses contour tracking [CCL04]. The 2D version of the algorithm is explained first and extended to 3D afterward. In the Algorithm 3.2, the procedure is also described in pseudo code. The data is iterated (from top to bottom, from left to right and in case of a 3D volume from front to back) for an unlabeled non-background pixel/voxel (let this pixel/voxel be the start pixel/voxel). If the above neighbor pixel/voxel $(j-1)$ is part of the background, this pixel/voxel is defined as an outer contour element. The contour is followed clockwise by selecting the first clockwise non-background element in the neighborhood. All elements of the contour are labeled. If the element below $(j+1)$ belongs to the background,this pixel/voxel is part of an inner contour. This contour is followed as well and all elements are marked with the current label. To mark the contours as processed, all background pixels/voxels between the entry pixel/voxel and the next contour pixel/voxel are set to -1. If none of these two cases are fulfilled, the left pixel/voxel $(i-1)$ has to be labeled and the current pixel/voxel gets its label from them. In 3D, also the $k-1$ indexed neighborhood is checked for the label number. The other steps are handled as the 2D version on the same k indices.

Algorithm 3.1: A Connected Component Analysis (CCA) method that scan the input data once and put unlabeled foreground elements in a queue to determine the neighbors that belongs to the same label.

Input: A binary image or volume (*inData*).
Output: An image or volume with labeled foreground elements (*outData*).

```
1  outData ← new empty image/volume with dimensions of inData;
2  currentLabel = 1 ;
3  q ← new empty queue ;
4  for element ∈ inData do
5      if element == foreground and element is unlabeled in outData then
6          set label in outData to currentLabel;
7          add element to q;
8          while q not empty do
9              first ← first element of q;
10             foreach neighbor of first in inData do
11                 if neighbor == foreground and neighbor is unlabeled in outData
                    then
12                     set label in outData to currentLabel;
13                     add neighbor to q;
14                 end
15             end
16             remove first from q;
17         end
18         currentLabel = currentLabel + 1;
19     end
20 end
21 return outData;
```

The above two algorithms label one object at a time. Another big group of CCA techniques uses two passes to assign the labels to the pixels based on the Hoshen–Kopelman algorithm [HK76]. In the first pass, all foreground-elements get a temporary label. If there are already labeled pixels in the neighborhood, the actual pixel gets the minimum of all labels or a new one if not. If there are different label numbers in the neighborhood, all numbers are stored in a list, because they can be merged. The second iteration is the merging step. The constructed merging list is used to unify all label numbers of the same component. This method can be optimized by using masks for the first iteration step [HCS12]. This masks contains foreground-background configurations to determine the labels of the first scan. This way it is not necessary to check all neighbors at every pixel/voxel and the runtime improves. There are also multipass or raster-scan approaches to solve the CCA problem.

Algorithm 3.2: A Connected Component Analysis (CCA) method that scan the input data once and uses contour tracking to determine the labels of the objects.

Input: A binary image or volume (*inData*).
Output: An image or volume with labeled foreground elements (*outData*).

```
1  outData ← new empty image/volume with dimensions of inData;
2  currentLabel = 1 ;
3  for element ∈ inData do
4      if element == foreground and element is unlabeled in outData then
5          if above neighbor of element belongs to the background then
6              track the contour and label it in outData to currentLabel;
7              mark background neighbors outside the contour with −1 in outData;
8          end
9          if below neighbor of element belongs to the background then
10             track the contour and label it in outData to currentLabel;
11             mark background neighbors inside the contour with −1 in outData;
12         end
13         if above and below neighbor of element are non-background then
14             label element in outData to the label of the left neighbor;
15         end
16         currentLabel = currentLabel + 1;
17     end
18 end
19 return outData;
```

Resolution Reduction

The runtime of most of the operators (filtering, CCA, feature extraction, ...) applied to 3D datasets depends on the given resolution. Decreasing the resolution would speed up the calculations directly [DGBCNÁVA17]. The challenge in this field is to preserve as much information as possible by simplifying the data. Let $\mathbf{D}$ be the $n \times m \times l$ 3D data matrix whose elements are $\mathbf{D}(i, j, k)(i \in [0, n-1], j \in [0, m-1], k \in [0, l-1])$. To get the sampling positions for the values of the downsampled matrix $\mathbf{D}'$ of resolution $m' \times n' \times l'$, the formula in Equation 3.7 can be used:

$$\mathbf{D}'(i', j', k') = \mathbf{D}(\frac{m}{m'} \cdot i', \frac{n}{n'} \cdot j', \frac{l}{l'} \cdot k'). \tag{3.7}$$

Because of the division, the new indices could be fraction numbers, there have to be some interpolation methods to determine the right integer indices for the original data. Due to the three-dimensionality, there will be eight possible values which have to be taken into account. The easiest way to decide the new value is using the nearest neighbor by rounding the indices to the next integer value. Another ways are using the mean value of the eight elements or the trilinear interpolation (more details in Section 3.8).

The results can be improved by preparing the source data. For example, a Gaussian filter would remove high frequencies in the data and the new sampled value would be more visually accurate. This filter calculates new values by taking a weighted average of the surrounding entries. More complex algorithms take the difference of the Gaussian filtered dataset with the original dataset as filter kernels to determine the new values [DGBCNÁVA17] or use sequences of similar values within the image/volume to compress the data to a smaller resolution [DFB19].

3.3.3 Principal Component Analysis

The Principle Component Analysis (PCA) is a method to get information about a given data. It has different outputs, which are the Eigenvectors and Eigenvalues (one for each dimension or feature of the data) and a transformation matrix to convert the original data points into the new Eigenspace [JC16]. As shown in Figure 3.8, the Eigenvectors describe significant directions of each dimension within the data (e.g., the first Eigenvector is the direction with the highest variance). The Eigenvectors are all orthogonal to each other. The Eigenvalues are these variances. If we sort the Eigenvalues in descending order, it is possible to leave out dimensions with lower values to reduce the dimensionality of the data without losing much relevant information. This helps to get a better overview of the data. In the following, the calculation of these values is described.

Let $\mathbf{E}$ be a $sc \times fc$ data matrix, with sc samples and fc features. If E corresponds to a grayscale image, the number of samples would be the numbers of pixels and there would be only the grayvalue as single feature. In case of numerical data, the features would be the extracted characteristics per sample. In the first step, the mean values of each column are calculated.

$$\mathbf{u}[j] = \frac{1}{sc} \sum_{i=0}^{sc-1} \mathbf{E}[i, j] \tag{3.8}$$

Formula 3.8 calculates a vector $\mathbf{u}$ of length $fc - 1$ which contains these mean values. The variable j iterates in the range $[0, fc - 1]$. Next (Equation 3.9), the data is centered by subtracting the obtained mean values with the original data.

$$\mathbf{Cent} = \mathbf{E} - \mathbf{t} \cdot \mathbf{u}^T \tag{3.9}$$

For the subtraction, the vector $\mathbf{t}$ is used, which has a length of sc and contains only ones. Multiplied with the transposed $\mathbf{u}$ vector, a matrix is obtained which can be subtracted from the data matrix. Now the covariance matrix **Cov** of **Cent** is needed. This matrix with a size of $fc \times fc$ contains in the diagonal elements the variances (i.e., covariance of the element with itself) and in the others the covariances between feature pairs.

$$\mathbf{Cov} = \frac{1}{sc - 1} \cdot \mathbf{Cent}^H \cdot \mathbf{C} \tag{3.10}$$

As Equation 3.10 shows, a multiplication of **Cent** with its conjugate transpose ($\mathbf{Cent}^H$) has to be done. If **Cent** contains only real numbers (in this thesis always the case), the

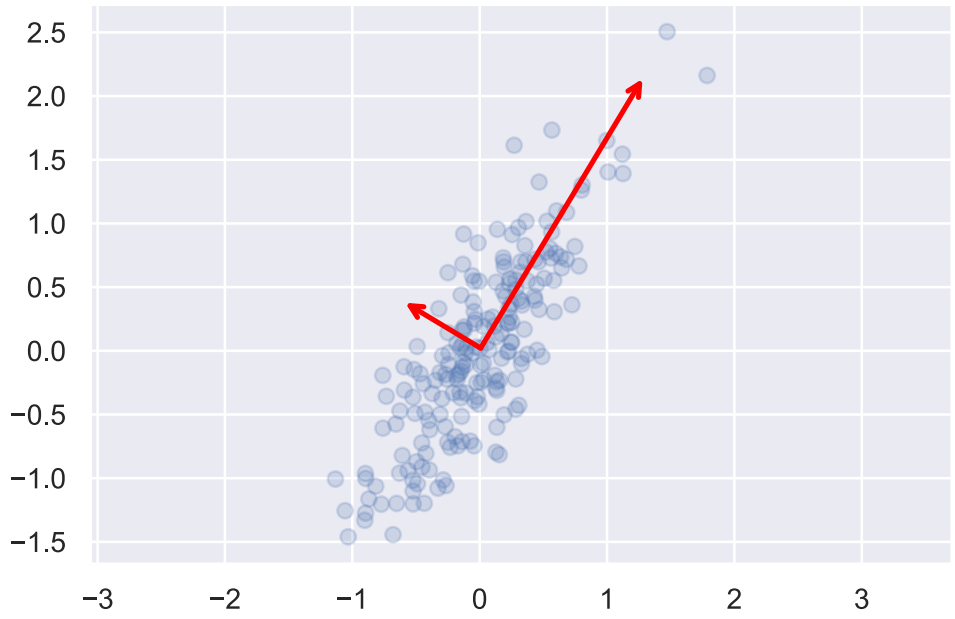


Figure 3.8: Example of Principle Component Analysis (PCA) output. The data is analyzed for its main directions (visualized in the direction of the red lines) and the amount of the variances (length of the red lines) namely the Eigenvectors and Eigenvalues.

transposed matrix of **Cent** can also be used, because the conjugation does not affect the result. In the final step, two matrices are searched which fulfill the equation defined in Formula 3.11:

$$\mathbf{V}^{-1} \cdot \mathbf{Cov} \cdot \mathbf{V} = \mathbf{K}. \tag{3.11}$$

K is a diagonalized $fc \times fc$ matrix whose diagonal elements contain the Eigenvalues of **Cov**. The Matrix **V** (also of dimension $fc \times fc$) contains the Eigenvectors as its column vectors. For the Eigenvalues, the characteristic polynomial $p_A(\lambda)$ has to be found. The definition of the polynomial is described in Equation 3.12:

$$p_A(\lambda) = det(\mathbf{Cov} - \lambda \cdot \mathbf{I}). \tag{3.12}$$

The elements in this equation are the following: **I** s the identity matrix (ones in the diagonal, zero elsewhere), λ iis the variable of the polynomial and *det* is the determinant of the matrix argument. With $p_A(\lambda)$, the Eigenvalues can be obtained by setting the equation to zero and solving for lambda. The Eigenvectors can be obtained similarly. For each Eigenvalue λ_i (i iterates from zero to the number of Eigenvalues) Equation 3.13 has to be solved:

$$(\mathbf{Cov} - \lambda_i \cdot \mathbf{I}) \cdot \mathbf{v_i} = 0, \tag{3.13}$$

with $\mathbf{v_i}$ being the Eigenvector of λ_i of length fc. The Matrix $\mathbf{V}$ (Equation3.11) is a $fc \times fc$ matrix which columns are the Eigenvectors.

As the Eigenvectors are all orthogonal to each other, it is possible to transform the given data into the Eigenspace and use the dimensions with the highest variances (i.e., biggest Eigenvalues) as the new axes. If this is done, the dimensions loose their initial meanings, but it is possible to visualize the components of the data with the highest variances. This helps to reduce dimensions for a better analysis of the data without losing significant parts of the reduced information. The transformation is being done with a left side matrix multiplication of the transposed Data Matrix $\mathbf{E}$ and the inverse of the Eigenvectormatrix $\mathbf{V}$ (Equation 3.14):

$$\mathbf{E}_{Eigenspace} = \mathbf{V}^{-1} \cdot \mathbf{E}^T. \tag{3.14}$$

In the visualization part of this thesis, only the first two rows of $\mathbf{E}_{Eigenspace}$ are taken into account, because the first two dimensions have the highest variances. This results in a $2 \times sc$ matrix in which every column represents a sample point.

The PCA output can also be used to achieve a fast approximate registration of the seeds that is sufficient for the visual purposes of this work. For this, the PCA is calculated over the input volume data which is, in fact, a 3D matrix. The elements in this matrix are density values. The easiest way to prepare the 3D matrix for the PCA input is, to store all voxel indices belonging to a contour or to the foreground. sc is the number of these voxels, fc is three and the features are the indices. It is possible to calculate the PCA and get the (three) main directions of the density values. Now it is possible to compute a rotation matrix which transforms the directions of the volume (calculation shows below). This rotation matrix can be used in a volume rendering step to get an equal rotation of every volume. This kind of registration is not exact and sometimes not unique, because the seeds are rotationally symmetric in at least one main direction. It is possible, that some of the seeds are rotated by 180°, because of their symmetric contours.

A fast method to calculate the rotation matrix mentioned above is the Rodrigues' rotation formula [Meb07]. This technique tries to minimize the use of costly functions like sine, cosine, or square root. Let $\mathbf{a}$ and $\mathbf{b}$ be two normalized 3D vectors and $\mathbf{R}$ be the searched rotation matrix transforming a to b. The calculation formula of $\mathbf{R}$ is shown in Equation 3.15:

$$\mathbf{R} = \mathbf{I} + \mathbf{P} + \mathbf{P}^2 \cdot \frac{1}{1+c}, \tag{3.15}$$

with $\mathbf{I}$ being the identity matrix, $\mathbf{P}$ is a helper matrix (the elements are defined in Equation 3.16) and $c = \mathbf{a} \cdot \mathbf{b}$ (dot product). Normally, the term $\frac{1-c}{s^2}$ is used at the end (with $s = ||\mathbf{a} \times \mathbf{b}||$), but it can be rearranged to the used term ($\frac{1}{1+c}$).

$$\mathbf{P} = \begin{bmatrix} 0 & -p_3 & p_2 \\ p_3 & 0 & -p_1 \\ -p_2 & p_1 & 0 \end{bmatrix} \tag{3.16}$$

p_i ($i \in 1, 2, 3$) are the elements of $\mathbf{p} = \mathbf{a} \times \mathbf{b}$. Programmatically , there are two special cases which should be handled separately. The first one is $\mathbf{a} = \mathbf{b}$, then $\mathbf{R} = \mathbf{I}$ and the second one is $\mathbf{a} = -\mathbf{b}$, then the rotation is simply 180 degrees.

Thus, there are three dimensions to rotate, three rotation matrices are needed (one for each Eigenvector of the volume). These three matrices can be multiplied with each other. As matrix multiplication is not commutative, it is important to retain the multiplication order for each seed, which should be registered, to get similar rotations. In case of this work, the seeds are all rotated axis aligned, the rotation matrix contains simply the inverse Eigenvectors which is $\mathbf{V}^{-1}$

3.4 Features

In case of the CT data analysis, features are numerical properties of the scanned objects extracted with different calculations. These features can be used to get the measurements of the scanned physical objects (e.g., length, width, volume) for comparison, description, or recognition purposes. They can be subdivided into geometrical shape features and texture features of the surface structure. Most of the, in the following described, features can be calculated for both, two-dimensional images (like slice data) or three-dimensional volumes. In this work, the 3D versions of the feature calculations are of interest.

3.4.1 Shape Features

This section illustrates features, which depend only on the physical appearance of the data and not of abstractions of it. Since the raw volumes are not aligned with each other, it is important to have rotation invariant characteristics. For this, the principal axes (i.e., Eigenvectors) are calculated in the first place. The specific technique for the calculation is described in Section 3.3.3. Due to the principal axes the orientation of the object within the scan can be ignored. They yield information on the major directions. In case of 3D, there are three of them. The direction with the shortest magnitude is called the minimal, the longest one the maximal and the one between them is the intermediate principal axis. In the following, common shape features and their calculation are described.

- Elongation: This feature is defined as the length of the intermediate principal axis divided by the length of the maximal axis [ZW16].
- Flatness: The flatness is similar defined as the elongation, only the used axes differ. This value is calculated as the quotient of the minimal and the intermediate axis [ZW16]. It is also possible to represent the flatness as a measurement for how planar or abrasive the surface of a shape is [Yan95].
- Perimeter/Surface Area: For the calculation of objects with more than two dimensions, the Crofton formula is often used [LL12]. It says that the length of a line is

the sum of its points, so an approximation of the perimeter/surface area can be calculated by counting all points on the contour of an object. The main idea is, to lay a grid over the object and sum up all intersections between the contour of the object and the lines of the grid. If the distance between the gridlines is infinitely small, the exact perimeter can be obtained. Since a smaller distance between the grid lines than the resolution of the 3D volume is not necessary, the resolution of the object can be used as discretization, and the formula of the perimeter/surface $Per(X)$ can be simplified to Equation 3.17:

$$Per(X) \approx 4 \cdot \sum_{dir} \frac{c_{dir}}{step_{dir}} \cdot \chi(X \cap L_{dir}), \tag{3.17}$$

where X is the input object and dir iterates over the grid directions (in case of 3D there are three directions but additional diagonal grid lines are also possible). The following parameters are all with respect to their direction dir. c_{dir} is a discretization weight ($c_{dir} = \frac{1}{3}$ if only the 3D main directions x, y and z are used) to get rid of multiple counted points. $step_{dir}$ is the density of the grid lines (the distance between two voxels in direction dir divided by the size of the voxels) and L_{dir} is the set of the discrete lines in direction dir of the object. χ is the Euler-Poincaré characteristic which is equal to half the number of intersections of the boundary of the object X with the line L_{dir}.

- Equivalent Spherical Radius: This characteristic contains the radius R_{eq} from a perfectly spherical shape with the same volume V as a target object with dimensionality d [LL12]. The equations needed for the calculation are described in Equation 3.18 to 3.20.

$$R_{eq} = \sqrt[d]{\frac{V \cdot \Gamma(\frac{d+1}{2})}{\pi^{\frac{d}{2}}}} \tag{3.18}$$

$$\Gamma(\frac{d+1}{2}) = \begin{cases} \frac{d}{2}! & \text{if d is even} \\ \frac{\sqrt{\pi} \cdot d!!}{2 \cdot \frac{d+1}{2}} & \text{if d is odd} \end{cases} \tag{3.19}$$

$$d!! = \begin{cases} 1 & \text{if } d \leq 1 \\ d \cdot (d-2)!! & \text{otherwise} \end{cases} \tag{3.20}$$

Equation 3.19 describes the Eulerian Gamma function and Equation 3.20, the double factorial. In case of 3D, this formula results into Equation 3.21:

$$R_{eq} = \sqrt[3]{\frac{V \cdot 3}{\pi \cdot 4}} \tag{3.21}$$

- Equivalent Spherical Perimeter: This value describes the perimeter/surface area of a d dimensional object with the same volume V as the shape of interest [LL12]. The calculation is shown in Equation 3.22:

$$P_{eq} = \frac{d \cdot V}{R_{eq}}, \tag{3.22}$$

where R_{eq} is the equivalent spherical radius.

- Roundness: This feature is a value in the range $[0, 1]$, where one describes a perfectly spherical object and which decreases for more complex structures [LL12]. Generally, it is defined as the ratio between the two perimeters or surface areas, one of a spherical object with the same volume/area P_{eq} and one of the searched object P. For 3D structures, it is also possible to take the ratio between the surface area and the volume as an alternative interpretation of the roundness.

- Volume: In discrete data, the volume is equivalent to the number of voxel within the object boundary. Is the physical size of one voxel known, it is possible to estimate the volume of the real subject of interest (i.e., seed).

- Principal Moments: The principal moments are the moment of inertia I_{dir} (other names: angular mass or rotational inertia) respective to the principal axes. They describe the force needed to rotate the object in the desired direction [MO11]. In general, this feature can be calculated in arbitrary direction, but for the comparability purpose, it is necessary to have fixed axes for every object. It is calculated as the sum of the product of all point distances to the rotation axis and their masses (Equation 3.23). If the masses of the voxels are unknown, they can be set to a constant value or a function depending on the volume.

$$I_{dir} = \sum_{i=0}^{vc-1} m_i \cdot r_i^2, \tag{3.23}$$

  where dir is the direction, vc is the number of voxels within the object, m_i is the mass of the point i and r_i is the distance (e.g., Euclidean) of the point to the rotation axis. The sum iterates across all points of the object. This moment can be observed by figure skaters while doing a pirouette. With extended arms, the moment of inertia is bigger and the rotation slower. Are the arms held tightly on their sides, it is smaller and the rotation faster.

- Circularity index: This index C_{dir} can be calculated for each dimension and shows the similarity with a circle [CMS16] (Equation 3.24). In 3D, cross-sections are used and a circularity index for each direction is calculated.

$$C_{dir} = \frac{4 \cdot \pi \cdot area}{perimeter^2} \tag{3.24}$$

  In principal, every cross-section in direction dir of the volume can be used to calculate the $area$ and the $perimeter$. To get a feature that can be compared between different objects, the direction can be set to the principal axes and the used cross section can be the objects midpoint in the respective direction.

- Rugosity: Is the ratio between the surface area of the seed and the surface area of the convex hull of the seed [CMS16].

- Equivalent Ellipsoid Diameters: Represents, similar to the spherical case, the diameters of an ellipsoid with equal volume. In 3D there are three diameters along the principal axes.

3.4.2 Texture Features

A helpful tool to gain texture information (distribution of grayvalues within the data volume or image (i.e., slice data)) is the Grey Level Co-occurrence Matrix (GLCM) [BJT08]. The elements $\mathbf{Glc}(i,j)$ of this grey level matrix **Glc** of size $gv \times gv$ (where gv is the number of grayvalues) contains the number of gray level transitions within the image from gray value i to grayvalue j. A gray level transition is a pair of neighboring pixel/voxel gray values (e.g., the grayvalues of a pixel/voxel and its right neighbor). If for example $\mathbf{Glc}(10,200) = 15$, there are 15 gray level transitions from 10 (light gray) to 200 (dark gray). The elements are normalized by dividing them by the number of all pixels. **Glc** can be seen as gray level transition histogram. If this matrix is created, the following texture features can be calculated.

- Angular Second Moment (ASM): The smoothness of the texture is measured by the sum of squared entries of **P** (Equation 3.25):

$$ASM = \sum_{i=0}^{gv-1} \sum_{j=0}^{gv-1} \mathbf{Glc}(i,j)^2 \tag{3.25}$$

- Contrast (CON): The contrast is a measurement on how equally distributed the greyvalues are (Equation 3.26):

$$CON = \sum_{i=0}^{gv-1} \sum_{j=0}^{gv-1} \mathbf{Glc}(i,j) \cdot (i-j)^2 \tag{3.26}$$

- Dissimilarity (DIS): This feature is like the contrast but with linear weights. It results also in larger values for higher contrast within the source volume/image of Glc (Equation 3.27):

$$DIS = \sum_{i=0}^{gv-1} \sum_{j=0}^{gv-1} \mathbf{Glc}(i,j) \cdot |i-j| \tag{3.27}$$

- Homogeneity (HOM): Higher values of this feature describe more homogenous textures (Equation 3.28):

$$HOM = \sum_{i=0}^{gv-1} \sum_{j=0}^{gv-1} \frac{\mathbf{Glc}(i,j)}{1+(i-j)^2} \tag{3.28}$$

- Maximum Probability (MAX): Returns the highest transition probability of the source volume/image of *Glc* (Equation 3.29):

$$MAX = max(\mathbf{Glc}(i,j)) \tag{3.29}$$

- Entropy (ENT): Is a measurement of disorder (Equation 3.30). The rule $0 \cdot ln(0) = 0$ is used to get proper results if $\mathbf{Glc}(i,j)$ is null:

$$ENT = \sum_{i=0}^{gv-1} \sum_{j=0}^{gv-1} \mathbf{Glc}(i,j) \cdot (-ln(\mathbf{Glc}(i,j))) \tag{3.30}$$

- Energy (ENG): This feature measures the orderliness (Equation 3.31):

$$ENG = \sqrt{ASM} \tag{3.31}$$

- Correlation (COR): Shows the linear dependency of the gray values respective to their neighbors (Equation 3.32):

$$COR = \sum_{i=0}^{gv-1} \sum_{j=0}^{gv-1} \mathbf{Glc}(i,j) \cdot \frac{(i-\mu_i) \cdot (j-\mu_j)}{\sqrt{\sigma_i^2 \cdot \sigma_j^2}}, \tag{3.32}$$

  where μ_i and μ_j are the mean values in i and j directions calculated with Equation 3.33. σ_i and σ_j are the respective variances (Equation 3.34).

$$\mu_i = \sum_{i=0}^{gv-1} \sum_{j=0}^{gv-1} i \cdot \mathbf{Glc}(i,j) \quad , \quad \mu_j = \sum_{i=0}^{gv-1} \sum_{j=0}^{gv-1} j \cdot \mathbf{Glc}(i,j) \tag{3.33}$$

$$\sigma_i^2 = \sum_{i=0}^{gv-1} \sum_{j=0}^{gv-1} \mathbf{Glc}(i,j) \cdot (i-\mu_i)^2 \quad , \quad \sigma_j^2 = \sum_{i=0}^{gv-1} \sum_{j=0}^{gv-1} \mathbf{Glc}(i,j) \cdot (j-\mu_j)^2 \tag{3.34}$$

3.5 Convex Hull

An important task of this work is to detect outliers within the features of the data. The convex hull describes the smallest closed polygon which encloses all points of a given set. It has a big range of applications in Computer Graphics, like for collision detections [LZB08], boundary analyses [WBP+16], bagplots [RRT99] and many more. In this section, an algorithm is described which can handle the calculation in linearithmic ($O(n \cdot log(n))$) runtime, because of real-time interactions the results should be available as fast as possible. Since a 2D scatter plot will be used, the method will handle sets of 2D points.

The method described is the Graham algorithm [GY83]. This technique uses the following property of the convex hull: there is no turn right by tracking on the polygon of the convex hull counterclockwise. Let $S = \{\mathbf{point}\}$ be a set of $2D$ points. The ensuing three steps illustrate the procedure of the method.

- Define the first point as the one with the minimal y coordinate. If there are more than one, take the point with the lowest x coordinate. This point $\mathbf{point_0}$ lies definitely on the convex hull and serves as the reference point.
- Order the other points $\mathbf{point_i}$ based on their angle between a baseline (i.e., horizontal line) through the reference point and the line connecting $\mathbf{point_0}$ and $\mathbf{point_i}$. Take the point farthest away from $\mathbf{point_0}$ if there are equal angles.
- Check for every three following points in the list sorted in ascending order, if there is a turn right, the middle point is not part of the convex hull.

Now we have a more detailed look at each step. The first one is trivially finding the lowest point $\mathbf{point_0}$ in the set S. In the second step, the ordering through all points can be done with Equation 3.35:

$$angle = acos(\frac{x_{dist_i}}{||\mathbf{dist_i}||}) \tag{3.35}$$

where $\mathbf{dist_i} = \mathbf{point_i} - \mathbf{point_0}$ is the direction from $\mathbf{point_0}$ to $\mathbf{point_i}$. Since the searched angle is, in fact, the angle between $\mathbf{dist_i}$ and the x axis, the numerator of this ratio (normally the dot product $\mathbf{dist_i}$ and the x axis) can be simplified to the x coordinate of the direction vector $\mathbf{dist_i}$. If multiple points have the same angle, the farthermost point is chosen. The main step of the algorithm checks the rotational direction for every three consecutive points in the sorted list. This can be done by comparing the slopes of the two connection lines between $\mathbf{point_i}, \mathbf{point_j}$ and $\mathbf{point_j}, \mathbf{point_k}$ [FTU95]. Equation 3.36 shows the calculation for the comparison values:

$$\begin{aligned} m_{i,j} &= \frac{y_{point_j} - y_{point_i}}{x_{point_j} - x_{point_i}} \\ m_{j,k} &= \frac{y_{point_k} - y_{point_j}}{x_{point_k} - x_{point_j}}. \end{aligned} \tag{3.36}$$

If $m_{i,j} > m_{j,k}$, there is a right turn otherwise it is a left turn. To avoid divisions it is possible to rearrange this formula to get Equation 3.37:

$$m = (y_{point_j} - y_{point_i}) \cdot (x_{point_k} - x_{point_j}) - (y_{point_k} - y_{point_j}) \cdot (x_{point_j} - x_{point_i}). \tag{3.37}$$

The orientation check changes to $m > 0$. If this expression is true, there is a right turn and $point_j$ is not part of the convex hull. The easiest way to implement the presented Graham algorithm is to use a stack in the third step. Initially, place the first two elements of the sorted list in the stack. Then always test the two top entries of the stack and the next element of the ordered list. In case of right turn, pop the top element of the stack otherwise push the element of the ordered list onto the stack. This algorithm is also shown in Algorithm 3.3 in pseudocode.

Algorithm 3.3: Calculate Convex Hull

Input: A list of 2D points (*points*)
Output: A list of 2D points which represents the convex hull polygon (*outPolygon*).

create an empty list *outPolygon*;
bottomMostPoint ← bottom most point of *points* . If there are multiple one, select the left most of them;
swap *bottomMostPoint* with the first position of the *points* list;
sort all elements of *points* but *points*[0] respective to their polar angle to *points*[0] (counterclockwise)
$(orientationofthreepoints = (point1.y - point0.y) \cdot (point2.x - point1.x) - (point1.x - point0.x) \cdot (point2.y - point1.y))$;
remove all elements of *points* of same angle, but the farthest away. Let the new number of elements be m;
if *remaining elements of points* < 3 **then**
    no convex hull possible;
end
create a stack element (*stack*) and fill it with the first three elements of *points* (note: a modified stack is used, where also the element after the top can be accessed);
for $i \leftarrow 3$ **to** m **do**
    while *orientation of stack.oneBeforeTop, stack.top and points*[i] *is a right turn* **do**
        remove top element of the stack;
    end
    add *points*[i] on the stack;
end
put all elements of the stack into *outPolygon*;
return *outPolygon;*

3.6 Kernel Density Estimation

For the domain expert, it is interesting to see the main location of the different species respectively to specific features. The Kernel Density Estimation (KDE) (or Parzen–Rosenblatt Window method) is a method to determine the probability density function behind a random variable [Par62]. Value regions of a feature which contain many samples will get a higher probability than value regions with fewer elements. The resulting function can be visualized for a better overview of the data. Let **data** be a data vector of size dc, then the Equation 3.38 shows the basic function of the estimation:

$$f(x) = \frac{1}{dc \cdot h} \cdot \sum_{i=0}^{dc-1} K(\frac{x - \mathbf{data}_i}{h}), \tag{3.38}$$

where h is the size of the kernel and K is a probability measurement function. The function K has to be satisfying $\int K(x)dx = 1$. Some example kernels can be found in the list below [Węg18].

- Gaussian ($\sigma = 1$): $K(x) = \frac{1}{\sqrt{2 \cdot \pi}} \cdot e^{-\frac{x^2}{2}}$
- Epanechnikov: $K(x) = \begin{cases} \frac{3}{4 \cdot \sqrt{5}} \cdot (1 - \frac{1}{5} \cdot x)^2 & \text{if } |x| < \sqrt{5} \\ 0 & \text{otherwise} \end{cases}$
- Biweight: $K(x) = \begin{cases} \frac{15}{16} \cdot (1 - x^2)^2 & \text{if } |x| < 1 \\ 0 & \text{otherwise} \end{cases}$
- Triangular: $K(x) = \begin{cases} 1 - |x| & \text{if } |x| < 1 \\ 0 & \text{otherwise} \end{cases}$

The critical parameter of the KDE is the bandwidth h. If h is too small, there will be high frequencies in the output. With an oversized value, the outcome will be too smoothed out. If the runtime is a critical issue, it is a good idea to set h to a fixed value. The correct bandwidth can only be calculated if the underlying probability function is known (which is mostly only the case if the data is synthetic). There are some approximation methods available which are trying to estimate the appropriate value [Sil86]. A commonly used technique is the rule-of-thumb method. The standard formula is shown in Equation 3.39:

$$h = 1.06 \cdot \sigma \cdot dc^{-\frac{1}{5}}, \tag{3.39}$$

where σ is the standard deviation (calculation shown in Equation 3.40) and dc is the number of data samples.

$$\sigma = \sqrt{\frac{1}{dc} \cdot \sum_{i=1}^{dc} (x_i - \mu)^2} \tag{3.40}$$

μ is the mean value of the data (i.e., sum of all points divided by dc). For more robustness against outliers, the so-called Inter Quartile Range (IQR) can be used in combination with an adjusted modifier (like in Equation 3.41).

$$h = 0.79 \cdot IQR \cdot dc^{-\frac{1}{5}} \tag{3.41}$$

A quartile is a specific element in a sorted sequence of values. The upper quartile is called $x_{0.75}$ and the lower quartile $x_{0.25}$. The IQR is defined as the difference between the upper ($x_{0.75}$) and the lower quartile($x_{0.25}$) of the data [UC96]. The median, for example, is also called the $x_{0.5}$ quartile. As for the median, the data samples are sorted in ascending order. The values of the quartiles are determined as shown in Equation 3.42.

$$\begin{aligned} x_{0.25} &= \begin{cases} \frac{1}{2} \cdot (x_{dc \cdot 0.25} + x_{dc \cdot 0.25+1}) & \text{if } dc \cdot 0.25 \text{ is an integer} \\ x_{\lfloor dc \cdot 0.25+1 \rfloor} & \text{otherwise} \end{cases} \\ x_{0.75} &= \begin{cases} \frac{1}{2} \cdot (x_{dc \cdot 0.75} + x_{dc \cdot 0.75+1}) & \text{if } dc \cdot 0.75 \text{ is an integer} \\ x_{\lfloor dc \cdot 0.75+1 \rfloor} & \text{otherwise} \end{cases} \end{aligned} \tag{3.42}$$

The rule in Equation 3.41 works fine for unimodal distributions. To not over-smooth non-unimodal distributions, Equation 3.41 can also be improved to get Equation 3.43 which is the so-called Silverman's bandwidth:

$$h = 0.9 \cdot min(\sigma, \frac{IQR}{1.34}) \cdot dc^{-\frac{1}{5}}. \tag{3.43}$$

The Scott rule-of-thumb is also an example of this kind of approximation [Sco79]. There are several other techniques to calculate the bandwidth which are more accurate at the expense of runtime. The bandwidth h can, for example, be optimized by a Poisson point process assumption [SS10], by cross-validation [Wu19], or by plug-in methods [TAS08].

In 2D the KDE works analogous. The input of the kernel function is a 2D vector **vec** and the kernel functions change a little bit. An example of the 2D Gaussian kernel is shown in Equation 3.44 [WJ93]:

$$K(\mathbf{vec}) = \frac{1}{\sqrt{2 \cdot \pi}} \cdot e^{-\frac{\mathbf{vec}^T \cdot \mathbf{vec}}{2}} \tag{3.44}$$

3.7 Classification

A task of this thesis is to classify the archaeological data to get a prediction for the unknown seeds. Since a ground truth for a lot of classes is available, supervised machine learning is the chosen technique. In supervised learning, a domain expert has pre-labeled some samples [KZP07]. These samples are the basis to train a classifier for further predictions. For further validations, the set of samples are split into a training and a testing set. A scheme of an example pipeline is shown in Figure 3.9

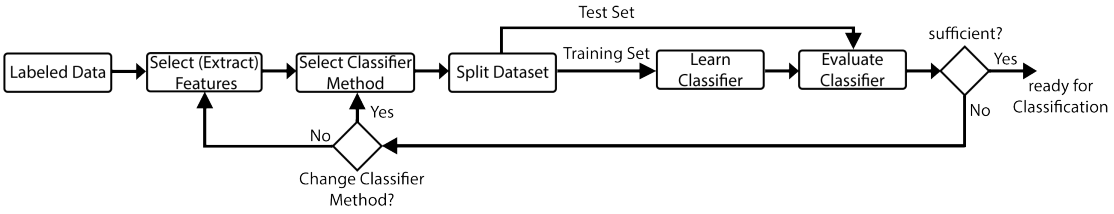


Figure 3.9: A scheme of an example pipeline for a supervised machine learner to create a classifier. The labeled features are split between a training and a test dataset. If the accuracy is to small, the classifier method or the used features can be changed for improving.

Since the task is to differentiate between various classes, only multiclass classifier methods are selected. The first classification technique described is the Gaussian Naive Bayes classifier, because it is one of the basic approaches in supervised machine learning [Zha04]. Naive Bayes classifiers are based on the Bayes' Theorem, which relates the conditional

probabilities of two random events. Let $X = (x_0, x_1, ..., x_{n-1})$ an unlabeled datapoint with $n-1$ features and $C = (c_0, c_1, ..., c_{m-1})$ a list of labels, then the probability of X having the label c_k with k is an integer in the set $0, 1, ..., m-1$ can be calculated with Equation 3.45:

$$p(c_k|X) = \frac{\prod_{i=0}^{n-1} p(x_i|c_k) \cdot p(c_k)}{\prod_{j=0}^{n-1} p(x_j)}. \tag{3.45}$$

$p(c_k)$ defines the probability of the class c_k which is, in fact, the relative frequency of the class within C. $p(x_i)$ is the equivalent relative frequency within X and $p(x_i|c_k)$ is the probability of getting x_i if c_k is true. Approximationg the probability $p(x_i|c_k)$ is the main difference between most of the techniques which use the Bayes Model and is also the part which is trained. In the case of the Gaussian method, $p(x_i|c_k)$ is assumed to be normally distributed. This means it can be calculated with Equation 3.46:

$$p(x_i|c_k) = \frac{1}{\sqrt{2 \cdot \pi} \cdot \sigma_{c_k}} \cdot e^{-\frac{(x_i - \mu_{c_k})^2}{2 \cdot \sigma_{c_k}^2}}, \tag{3.46}$$

$\sigma_{c_k}^2$ is the variance and μ_{c_k} is the mean of the trained elements in class c_k. The calculation of the variance (with equal probabilities for every element can be seen in Equation 3.47:

$$\sigma_{c_k}^2 = \frac{1}{l} \cdot \sum_{i=0}^{l-1} (c_k(i) - \mu_{c_k})^2, \tag{3.47}$$

where l is the number of elements in the class c_k and $c_k(i)$ is the i-th element of this class. In the training step of this classifier, the mean and the variances of each class of the training dataset is calculated and stored for further classifications. Another possibility to estimate $p(x_i|c_k)$ is the KDE with a Gaussian kernel to [RLC+09].

The next techniques described are the k-Nearest-Neighbor (kNN) and the Weighted k-Nearest-Neighbors (wkNN) classifiers [HS04]. The training step of these classifiers contain simply the storing of all training samples with their respective labels. New data samples are classified by taking the k nearest neighbors and looking up their labels. The label that appears most is the label of the new sample. For unique voting between two classes, the user-defined k should be an odd positive integer. Since in this thesis the probabilities of each class should be determined, getting only one class as output is not necessary. Larger values of k make this classifier more robust against noise but on the class boundaries the uncertainty increases. An example of this procedure is shown in Figure 3.10.

For the nearest neighbor calculation, a distance metric $dist(p_i, p_j)$ is used. The Euclidean distance or the absolute distance are examples of usable metrics. These metrics can be generalized to the so-called Minkowski distance shown in Equation 3.48:

$$dist(p_i, p_j) = \left(\sum_{f=1}^{n} |p_{i_f} - p_{j_f}|^q \right)^{\frac{1}{q}}. \tag{3.48}$$

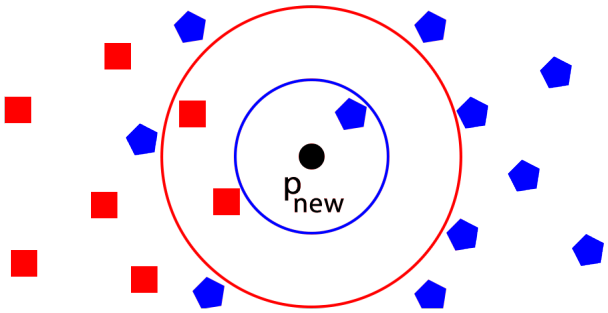


Figure 3.10: A scheme of the influence of the k parameter in the k-Nearest-Neighbor (kNN) technique. If $k = 1$, p_{new} will be assigned to the blue polygon class (blue circle). In the case of $k = 3$, it would be labeled as a red square class (red circle) member. $k = 5$ corresponds to the blue polygon class again.

p_{i_f} is the fth feature of the point p_i and p_{j_f} is the fth feature of the point p_j. The variable q set the kind of metric. For example $q = 2$ corresponds into the Euclidean distance. After the calculation of the distances between all points with respect to the new sample with unknown label, the k nearest ones are taken. Neighbor points with the same label are summed up to get the count of each label among these points. The count divided by k results in the probability of each class assigned to the new sample. If the values of the features are sorted, only the k elements around the feature values of the new sample have to be taken into account which safes some of the distance calculations.

To get rid of long-distance neighbors, the wkNN is used. In this technique, the distances of the k neighbors are converted into weights. This way, the label of neighbors that are far away gets a lower influence on the outcome. This is done by a kernel function (each kernel function in Section 3.6 can be used). For the kernel function a normalization has to be done. To standardize the distances between zero and one, they are divided with the distance to the $k + 1$th point which is definitely greater than all other distances. These normalized values are used as the x parameter in the kernel functions. The weights of samples with equal labels are summed up and the probabilities are the summed weights divided by the sum of all weights.

Another method is the Decision-tree classifier. In this classifier, decision rules based on the features are learned and saved [SL91]. The classification of new data is done by evaluating one rule after another. The results are the probabilities calculated from the proportion of the occurring classes of samples within the leaf node. An easy example to understand the evaluation of a decision tree would be the question: "Can we play outside?". The first rule could be "Is it sunny?", if yes then the algorithm return "yes". Otherwise the next rule "Does it rain?" is evaluated. If the answer is "no" the output is also "yes" and "no" otherwise. In the case of this thesis, the features are numerical data and there are more than two classes but the principle of evaluating one rule by another stays the same. The rules contain thresholds which split the input samples on a feature

into two subsets (based on the Classification and Regression Trees (CART) algorithm [RJPD14]). An example of a binary tree classification is shown in Figure 3.11.

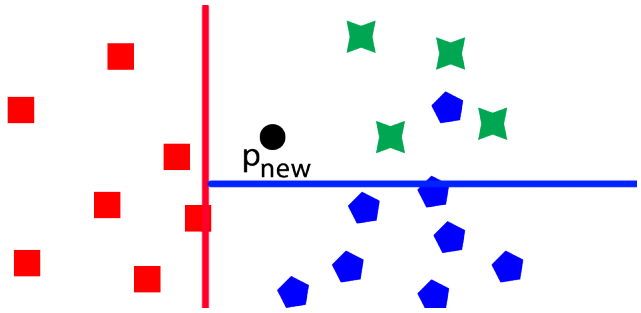


Figure 3.11: A scheme of a binary classification tree (e.g., Classification and Regression Tree (CART)) with tree classes and two features. In every rule a sample point within the data is used as a threshold to split the dataset into two subsets. Firstly the red rule and secondly the blue rule. The probability output of the new point would be 80% green diamond and 20% blue polygon class.

The threshold for the rules and the ordering of the features within the tree have to be found. This can be determined with the Gini Impurity criterion [Kan11]. The definition of this split criterion for an arbitrary set of data is shown in Equation 3.49:

$$Gini(D_{rc}) = 1 - \sum_{i=0}^{m-1} p(i_{rc})^2, \tag{3.49}$$

where D_{rc} are all samples of the rcth rule (i.e., D_0 would contain all samples, D_1 only a subset), m is the number of classes within D_{rc} and $p(i_{rc})$ is the fraction of the class i relative to all classes of D_{rc}. If the result of $Gini(D_{rc})$ is zero, all of the elements in D_{rc} belong to one class. To determine a perfect split into two subsets L_{rc} and R_{rc} where the union $L_{rc} \cup R_{rc} = D_{rc}$, a weighted Gini index is used (shown in Equation 3.50). Perfect split means that after the split only samples with the same label are in the two subsets.

$$wGini(D_{rc}) = p(L_{rc}) \cdot Gini(L_{rc}) + (1 - p(L_{rc})) \cdot Gini(R_{rc}). \tag{3.50}$$

$p(L_{rc})$ is the probability of a random sample selected in the left subset, which is the number of elements of the left subset divided by the number of all elements of D_{rc}. $Gini(L_{rc})$ and $Gini(R_{rc})$ are the $Gini$ indices of the respective subsets. For the final criterion value $g(D_{rc})$, the weighted index is subtracted by the Gini index of the whole set D_{rc} (Equation 3.51):

$$g(D_{rc}) = Gini(D_{rc}) - wGini(D_{rc}). \tag{3.51}$$

With these equations (Equation 3.49 to Equation 3.51), the best splitting threshold can be determined. The values of each feature of D_{rc} are put into different sets D_{rc_s} so

every set D_{rc_s} contains all elements of only one feature. The weighted Gini Impurance is calculated for each of these sets to calculate the split threshold. To determine the threshold for each set, every value within the sets are tried as threshold one by one. The threshold with the highest weighted Gini Impurance is taken. Now, $g(D_{rc_s})$ of each set is calculated with these thresholds and are compared to each other and the biggest one contains the feature used as a rule for this step rc. Now D_{rc} is split into R_{rc} and L_{rc} and these are the sets for the next tree iteration $rc + 1$. These calculations are repeated recursively for each subset until a stop criterion is fulfilled. Some example criteria are:

- maximum recursion depth
- maximum rule count
- minimum elements in subset
- Gini impurity of zero (i.e., all samples in set refer to the same class)
- Gini impurity higher than a threshold (i.e., only weak separation possible)

It is also possible that the same feature with another threshold is used as the best split. Besides the CART algorithm there are other techniques which use trees which are not restricted to binary branches, like the $C4.5$ algorithm [HMEE14]. This technique uses an Entropy and Information Gain approach instead of the Gini impurity (Equation 3.52 and Equation 3.53). 3.51).

$$Entropie(D_{rc}) = -\sum_{i=0}^{m-1} p_(i_{rc}) \cdot log_2(p(i_{rc})) \tag{3.52}$$

$$Gain(D_{rc}) = Entropie(D_{rc}) - p(L_{rc}) \cdot Entropie(L_{rc}) + (1 - p(L_{rc})) \cdot Entropie(R_{rc}) \tag{3.53}$$

These two parameters (the entropy and gain) are used to determine the threshold and the used feature of the rule in each step. A disadvantage of the tree classifiers is overfitting. If the sets are split into very small subsets with a lot of rules, it is possible, that the outcome gets worse than with lesser rules. Because of this behavior, a so-called pruning step is done after the tree creation. The classification error of the training set is calculated and stored. In the next step, the leave nodes are omitted one by one and the new error is determined. If the new error is less than the one of the whole tree, this reduced tree is used as the new decision tree.

A method which is very similar to the PCA but can be used as classifier is the Linear Discriminant Analysis (LDA) that can be used for classification and dimensionality reduction [Rao48] [LZO06]. The difference between them is, that the PCA finds axes which maximize the variance of the classes while the LDA calculates axes which have the best class separation. In general, five steps are performed. Let m be the number of classes and fc be the number of features. Additionally let x_i be a sample ($i \in [0, m-1]$)

and c_i the assigned class. The LDA starts by calculating a vector of dimension fc for each class which holds the mean value for each feature. In the next step, a within class and an in-between class matrix is created. A column of the within class matrix can be calculated with Equation 3.54:

$$\mathbf{S_w} = \sum_{i=0}^{m-1} \sum_{x \in c_i} (\mathbf{x} - \mu_{c_i}) \cdot (\mathbf{x} - \mu_{c_i})^T. \tag{3.54}$$

μ_{c_i} is the mean vector of class c_i. The in-between matrix elements can be calculated with Equation 3.55:

$$\mathbf{S_b} = \sum_{i=0}^{m-1} sc_i \cdot (\mu_{c_i} - \mu) \cdot (\mu_{c_i} - \mu)^T, \tag{3.55}$$

where μ is the overall mean and sc_i is the number of samples of class c_i. In the next step, the Eigenvectors and Eigenvalues of $\mathbf{S_w}^{-1}\mathbf{S_b}$ are calculated. Now, the Eigenvectors are sorted by decreasing Eigenvalues and a column matrix is created by concatenation of a predefined number of Eigenvectors. In the last step, the samples are transformed with this matrix into the new space by multiplication. The classification is, for example, done by distance measuring of the transformed new sample and the centroids of the classes.

The following classifiers are all part of the group of ensemble learners. These are classifiers who contains a set of base learners (sometimes called weak learners) and combine the predictions of all base learners for a final result [Zho09]. In principle, every machine learner can be used as a base learner. It is also possible to put different techniques into the ensemble and use them all. The ensemble results in most cases into more accurate predictions. Typically the learner is constructed in two steps. First, a constant number of base learners are generated (in parallel or sequentially). In the second step, the results of these learners are combined (mostly as a weighted average). A common method in ensemble learning is the Bootstrap Aggregation (Bagging) [Bre96]. In Bagging, each base learner is trained on a subset of the input training dataset which contains a number of randomly selected samples from the original set. The output is the probability distribution of the results from all base learners. A special case of Bagging is feature Bagging where the classifiers get a random set of the features they should use. Another ensemble technique is boosting [FSA99]. Different from bagging, boosting uses the base learners sequentially and each sample gets a weight. The first learner is trained with equal-weighted training samples and the classification error is calculated. The weights are modified in a way, that falsely classified samples get higher values afterwards. The next base learner uses the same training set with the new weights and so on. After every learner is trained, this procedure is repeated a predefined number of times. The weights are used to alter the probability values for a sample point assigned to a class. For example with k-Nearest-Neighbor (kNN) classifiers as base learner. If the output is 30% class one and 70% class two, but the samples of class two are weighted lower, the probability changes and the new sample may refer more likely to class one.

The first ensemble classifier described is the Random Forest Classifier [Ho95]. It is a standard bagging technique which uses a predefined number of decision tree classifiers as base learners. Each learner gets a random set of training data or features (or both) for learning. The voting of the learners can be weighted with their error rate.

The next ensemble described is the Adaboost Classifier which uses the method of boosting. This technique was originally introduced as a binary classifier [Sch13], but it can easily be extended to the multiclass case [HRZZ09]. The main challenge of every boosting algorithm is the calculation of the weights where w_i is the weight of the i^{th} element of the training dataset with tc samples ($i \in [0, tc]$). In the initialization step, all weights are set to $\frac{1}{tc}$. Let the error rate (err_{lc}) of a trained base learner on position lc be defined as in Equation 3.56:

$$err_{lc} = \sum_{i=0}^{tc-1} \frac{w_i \cdot C(x_i)}{\sum_{j=0}^{tc-1} w_j}. \tag{3.56}$$

$C(x_i)$ is a function which returns 1 if the sample x_i is misclassified (the predicted class is not the ground truth class) and 0 if x_i got the right prediction. A zero value of err_{lc} implies all samples are correctly classified, a false classification of all samples results in a value of one. With this error rate, the classifier weight α_m can be calculated as noted in Equation 3.57:

$$\alpha_{lc} = ln\left(\frac{1 - err_{lc}}{err_{lc}}\right). \tag{3.57}$$

α_{lc} is used to modify the vote of classifier lc and to refine the weight of each sample point of the training dataset. This way, samples of a learner with a high error get a lower weight. Equation 3.58 shows the formula for the new weights:

$$\bar{w_i} = w_i \cdot e^{\alpha_m \cdot C(x_i)}. \tag{3.58}$$

For predictions, the weighted sum of the output of every base learner is used. In this case, the weights of the classifiers α_{lc} are used. For multiclass predictions only the α_{lc} formula changes as shown in Equation 3.59:

$$\alpha_{lc} = ln\left(\frac{1 - err_{lc}}{err_{lc}}\right) + ln(m - 1). \tag{3.59}$$

One term, $ln(K - 1)$, is added where m is the number of different classes. In case of $m = 2$, the calculation for α_{lc} results in Equation 3.57

A different method which uses boosting is the Gradient Boost Classifier. Other than Adaboost, false classified samples are not marked with high weights, but with gradients [HTF09]. This gradients appear in a differentiable error function to fit the model on the underlying data. Since this approach makes only binary classification, there is one sub-classification for each class in the multiclass case. Every sub-classification decides if the new sample is within the class or not (with probabilities). The main part of this

method is the loss function $Loss(y_i, f(x_i))$. Where x_i is the i^{th} sample of the training set, c_i is the assigned class and $f(x_i)$ is the predicted class of x_i with i is an integer in the range $[0, sc-1]$. This function will be optimized during training to get the best predictions. There are many possible functions which can be used if they are differentiable. In the first place, the general approach is described which works for all suitable functions. In the second place, the specific likelihood loss function (which is commonly used) is given afterward as an example. The calculation of the initial prediction $f_0(x)$ is defined in Equation 3.60:

$$f_0(x) = \arg\min_{\gamma} \sum_{i=0}^{sc-1} L(c_i, \gamma). \tag{3.60}$$

So the γ is searched which minimizes the used loss function. A possible method to do this is, setting the derivation of the loss function to zero and solving for γ. c_i is zero if the sample x_i has not assigned the reviewed class and one otherwise (in the multiclass case, every class gets checked separately). The next steps are repeated in a loop a predefined number of times to refine this initial predictor. The pseudo residuals $r_{i,it}$ are calculated (Equation 3.61). These values contain information about how good the prediction is. i is the index of the sample and it the number of the current iteration.

$$r_{i,it} = -1 \cdot \frac{\partial L(c_i, f_{it-1}(x_i))}{\partial f(x_i)} \tag{3.61}$$

Like $f_{it-1}(x_i)$ notes, the predictions of the last step are used in the derivatives of the loss function. The residuals are stored for each sample in the training set, marked with the number of the current iteration it. In the next step, several features are selected to train a base learner (e.g., decision tree). This feature selection can be done randomly or with an algorithm which dies not take the same set of features for each iteration. The important part of this here is that the classifier is not learned with respect to the classes, but the pseudo residuals. A split is calculated on the selected features to determine which elements are classified as true (1) and which as false (0). For trees also which feature is used on which rule. Therefore the Gini index or the mean squared error (Equation 3.62) can be used:

$$err(S) = \sum_{i=0}^{sc-1} (r_{i,it} - \mu_S)^2, \tag{3.62}$$

where S is a subset of samples (e.g., the left or the right side of the split), sc is the number of samples in S, and μ_S is the mean value of the pseudo residuals in S. The feature value which minimizes $err(S)$ of the two partitions is then taken as threshold for the decision rule. The number of leaves T (classification endpoints) is typically a predefined property. Next, a new γ is needed for each leaf, to improve the classification of this iteration it. This is done with Equation 3.63:

$$\gamma_{j,it} = \arg\min_{\gamma} \sum_{x_i \in R_{i,j}} (L(c_i, f_{it-1}(x_i) + \gamma). \tag{3.63}$$

$R_{i,j}$ is the set of samples classified to the class of leaf j. j is an interger in the range of $[0, T-1]$). This minimization can also be done by setting the derivative to zero and

solving for γ. The last step of the loop is the calculation of the new probabilities for predictions $f_{it}(x)$ depended on the former prediction $f_{it-1}(x)$, which is $f_0(x)$ in the first loop (Equation 3.64) for each sample:

$$f_{it}(x) = f_{it-1}(x) + \nu \cdot \sum_{j=0}^{T_{it}-1} \gamma_{j,it} \cdot I(x \in R_{j,it}). \tag{3.64}$$

ν is the learning rate and controls the impact of a single iteration of the algorithm. It should be between 0 and 1 (typically it is near 0.1). The summation is only important if a single sample is classified into multiple leaves. Otherwise, it can be ignored. $\gamma_{j,it}$ is the result of the loss function from the leaf the sample is classified to. The last part, $I(x \in R_{j,it})$, is one if the sample is part of the set of samples of the leaf $R_{j,m}$ and zero else. it is set to $it + 1$ and the procedure is repeated until it reaches the predefined number of maximum iterations IT (typically 100). New classifications can be done by using $f_{IT}(p_{new})$.

A very common loss function is the log-likelihood function. This is a measurement for how good a statistical model fits to the underlying data. As a short recap: a gradient boost classifier has one classification part for every class, that only determine if a sample could refer to its class or not. For binary classification, the log-likelihood function LF is defined as shown in Equation 3.65:

$$LF = -1 \sum_{i=0}^{sc} c_i \cdot log(p) + (1 - c_i) \cdot log(1 - p) \tag{3.65}$$

It is the sum over all sc samples. c_i is the class of the sample (zero for false, one for true) and p is the probability that a random sample would classify to true. If $c_i = 1$ the second summand is 0 otherwise the first summand is ignored. At first p is set to $\frac{tsc}{sc}$. tsc is the number of samples assigned to the class. The factor -1 is important here because normally the high results in this function mean a better fit. In gradient boosting the loss function should be minimized. In this case, the formula has to be negated. The original likelihood function in Equation 3.65 contains also a summation over all samples of the set. Since the loss function will only handle one sample at a time, sc would be one and the summation can be ignored. This formula can be rewritten to Equation 3.66 by substitution:

$$LF = -c_i \cdot log(odds) + log(1 + e^{log(odds)}). \tag{3.66}$$

The calculation of $odds$ is shown in Equation 3.67:

$$odds = \frac{p}{1 - p}. \tag{3.67}$$

In some of the next steps, the derivative of LF is needed. This derivative is shown in Equation 3.68:

$$\frac{\partial LF}{\partial log(odds)} = -c_i + \frac{e^{log(odds)}}{1 + e^{log(odds)}} = -c_i + p. \tag{3.68}$$

For the initial prediction $f_0(x)$, the sum of the derivatives of all points is set to zero and solved for p (Equation 3.69):

$$\sum_{i=0}^{sc-1}(-c_i + p) = 0 \Rightarrow p = \frac{\sum_{i=0}^{sc-1}(c_i)}{sc}. \tag{3.69}$$

The sum is here necessary because the loss function has to be evaluated for each sample. Equation 3.69 is in fact $\frac{tsc}{sc}$, with tsc is the number of samples assigned to a class and sc is the number od samples. To get the log(odds), the Equation 3.67 is used and results in Equation 3.70:

$$log(\frac{\frac{1}{sc} \cdot \sum_{i=0}^{sc-1}(c_i)}{1 - \frac{1}{sc} \cdot \sum_{i=0}^{sc-1}(c_i)}). \tag{3.70}$$

Equation 3.70 can be understand as $log(\frac{tsc}{fsc})$ (tsc is the number of samples assigned to a class and fsc the number of samples not assigned to a class), which is the initial prediction of this loss function. The next step and simultaneously the first step of the loop of this method is the calculation of the pseudo residuals. With the log-likelihood, this is done with a simple subtraction (Equation 3.71):

$$r_{i,m} = c_i - f_{m-1}(x_i). \tag{3.71}$$

The pseudo residual of a sample x_i is the label of this sample c_i (one or zero) minus the prediction of the last iteration. Next, the base classifier is trained and the pseudo residuals are stored within the sample its belongs to. Let the name of the set of points within a terminal point (leave) of this classifier be $R_{j,m}$. A tree would have several terminal points, other learners will only have two of it. Now the new γ is searched for each $R_{j,m}$. Using the log-likelihood as loss function L, Equation 3.63 becomes to Equation 3.72:

$$\gamma_{j,m} = \frac{\sum_{x_i \in R_{j,m}} r_{i,m}}{\sum_{x_i \in R_{j,m}} f_{m-1}(x_i) \cdot (1 - f_{m-1}(x_i))}. \tag{3.72}$$

And finally, the new predictions are calculated as shown in Equation 3.64.

The next ensemble learner explained is the bagging classifier. As already noted, this kind of classifier uses its base learners in parallel and uses all results at the end [Bre96]. There are several properties to decide at the beginning of building such a classifier. First, the base learner used is decided. This can be simply a kNN classifier or a decision tree. Typically, one hundred of them are used in parallel but the number of learners is also variable. Second, the input samples and features are selected. It is possible to use random subsets of test samples for each learner, with replacement or without. This controls if a sample could appear in several random sets or if unique sets are used. Random sets of features can also be used in combination with the random samples or all samples are used in every base learner and only the used features differ. Finally, the composition of the results is setting up. This can be done by returning the average probability distributions of the outcomes (as so-called 'soft voting') or the maximum probabilities ('hard' or

'majority' voting). To suppress results of base learners with low accuracy in the 'hard' voting case, weights can be determined to rate the result of each classifier. The error rate of each classifier can be used to rate the prediction of each learner Bl_i (Equation 3.73):

$$err(Bl_i) = \frac{tsc}{sc}, \tag{3.73}$$

where tsc is the number of right classified samples and sc is the number of all samples. Since the sum of all error values should be one to get weights, they have to be normalized by dividing each with the sum of all error rates. Now the prediction probabilities of a sample with unknown label are the sum of the weights of the classifiers that predicts the same class. In soft voting, the probabilities for the same class of each learner is taken and the mean value is calculated for the resulting prediction.

The last method that has to be considered is the Convolutional Neural Network (CNN) [ZF14] or Artificial Neural Network (ANN) in general [BH00] which differs from the CNN mainly in the missing convolution layers. These networks are powerful machine learning tools which try to copy the human perception of objects. The whole system is build up in layers. At every layer, the features are getting more and more specific. The first layer may detect lines and edges, the second one detects rectangles and half circles out of the result of the first layer and so on. In the following, the different layers are described.

- Convolution Layer: This is the layer which gave the CNN its name. The convolution is done by applying a filter kernel on the input data. This works the same as the described techniques in Section 3.3.2. The filter kernel is shifted over the data and the result value is the sum of the element-wise products between the respective elements of the filter and the covered input data. The weights of the filters are mostly initialized randomly and getting adjusted step by step during the training process. The output size can be controlled by the step-size which controls the number of elements the filter skips during the shift operation. If the step-size is one, every element of the data will be operated. In case it is two, every second element is taken into account and so on. This step-size is called the stride.

- Rectified Linear Unit (ReLu) Layer: This layer introduces non-linearity into the network. It simply applies the function $f(x) = max(0, x)$ to each element of the input data. All negative values are set to zero because most of the real world data are non negative numbers. There are also other possible functions (like the tanh or sigmoid).

- Pooling Layer: This layer reduces the input data for better performance by removing less significant features. This step also uses a filter kernel. A specific function is applied to each element within the filter. Common functions are Max Pooling, Average Pooling, and Sum Pooling. The first one takes the maximum value, the second one calculates the mean value, and the last one sums up all values. These values are taken as output. Unlike the filtering which slides the kernel over each

element, this filter advances by the size of the filter, so every element is covered only once.

- Fully Connected Layer: The last layer converts the input matrix to an array and connects the elements of this array to an output array which has as many elements as there are classes to predict. Every class element has a connection to all elements of the input array. The output array is often normalized, so its sum is equal to one. This way, the values can be used directly as probabilities.

A CNN could contain these layers in the following pattern: Input → Convolution Layer → ReLu Layer → Pooling Layer → Convolution Layer → ReLu Layer → Fully Connected Layer. Many other combinations are available as well. If the accuracy of the network does not fit the expectations, more Convolution Layer → ReLu Layer combinations can be inserted. If a kernel does not fit perfectly into the layer resolution the padding can be done. The padding adds a defined number of zero columns and rows on each side of the data matrix to change the resolution. Also Features on the edges of the data are better preserved.

Now to the question, how such a network is trained. One method used is called back-propagation. The first step in back-propagation is a forward pass. A training sample is passed through the network with randomly initialized filters. In the next step a loss function L is used. This function could be for example the mean squared error which calculation is shown in Equation 3.74.

$$err = \frac{1}{sc} \cdot \sum_{i=0}^{sc-1} (c_i - f(x_i))^2 \tag{3.74}$$

sc being the number of samples, c_i the label of the sample x_i and $f(x_i)$ the predicted label of x_i. Next, in the backward pass, the layers are checked backwards to determine which weights contributed most to the loss. This works similar to the Gradient Boosting method. The weight is used which has the biggest impact on the loss function. In the last step, the weights are adjusted to the opposite direction of the gradient (Equation 3.75):

$$\bar{w_i} = w_i - \nu \cdot \frac{\partial L}{\partial w}. \tag{3.75}$$

w_i is the old weight, ν is the predefined learning rate and $\frac{\partial L}{\partial w}$ is the determined weight w which minimizes the loss function in this layer.

The described procedure is only a rough description. There are several more complicated networks available which use multiple convolution layers in parallel with different kernel sizes. An example is the GoogLeNet [SLJ+15] with 22 layers which contains multiple parallel filter kernels. CNNs are very accurate classifiers but need mostly a very high amount on training data for each class to gain its full power. Because most classes of this thesis have only view samples (e.g., four samples), it is assumed, that ANN are

too inaccurate on the given seed data. To validate this assumption one ANN is tested as a reference.

As a possible technique in the future, Active Shape Models (ASMs) [Coo00] and Active Appearance Models (AAMs) [CET01] [CET99] can be considered. These techniques use landmarks to create an average model from the samples. This model is fit to new samples with geometric transformations. A distance value between the new sample and the model is calculated and used for the classification. In this thesis, these models are not used because of the lack of user defined landmarks but it should be possible to use automatically generated landmarks on the surface of the seeds.

3.8 Volume Rendering

To visually inspect the CT scanned seeds, different methods should be used to offer different views on the data. The goal is, to use the containing density values of the scan to calculate a color to display on screen. In this section, the basics of Direct Volume Rendering are described. First, the principle of ray casting and approaches which can be used to detect the main values of a ray (startpoint/endpoint and direction) are explained. The different possibilities to compose the values on the ray are listed in the second part of this section.

In general, a ray is cast from each pixel of the screen and the displayed color is composed from all voxel values of the volume they hit. Let a ray be defined as a tuple $\mathbf{start}, \mathbf{end}$, where **start** is the start and **end** the endpoint of the ray. The direction **dir** of the ray can be simply calculated with these two points. All elements are vectors of size three, which is the dimensionality of a volumetric data. If **dir** is not normalized, $\mathbf{end} = \mathbf{start} + \mathbf{dir}$, $\mathbf{start} = \mathbf{end} - \mathbf{dir}$ and $\mathbf{dir} = \mathbf{end} - \mathbf{start}$. In some cases which are described later, a ray termination condition is available. If this condition is fulfilled, the ray terminates before the endpoint is reached and only the startpoint and the direction are from interest. Beside the ray, a sampling rate r is needed. This positive integer specifies the number of values on the ray which has to be considered in the calculation of the result value. Equation 3.76 shows how to iterate over a ray if all rays have equal length:

$$\mathbf{p} = \mathbf{start} + i \cdot (\mathbf{dir}/r), \tag{3.76}$$

where i is an integer in the range of $[0, r]$. If **dir** is normalized, Equation 3.77 can be used:

$$\mathbf{p} = \mathbf{start} + i \cdot \frac{||\mathbf{end} - \mathbf{start}||}{r} \cdot \mathbf{dir}. \tag{3.77}$$

Normally, the rays would not have equal length. In this general case and with a normalized direction dir, Equation 3.78 is used:

$$\mathbf{p} = \mathbf{start} + i \cdot \mathbf{dir}. \tag{3.78}$$

r has to be big enough to avoid staircase effects, but more samples increases also the runtime. **p** holds the coordinates to look up the values of the volumetric data. This data can be seen as a three-dimensional matrix **D** of size $n \times m \times l$. r has to be big enough, so no values of **D** are skipped but with higher values, the runtime increases without any improvement of the rendering. This is why r is normally defind by the resolution or voxel size of the data. $r = 2 * max(n, m, l)$ is a good approximation for the step size. Due to the division, the values of **p** will not be integer anymore. To overcome this issue there are several approaches. The straightforward way is the nearest neighbor interpolation. The values of **p** are only rounded to their respective integers. The positive effect of this method is, that only values are used, which are in the Data without any changes. In crucial topics, like medical visualization, this could be very important. Another possibility is trilinear interpolation. A new value is calculated from the surrounding eight elements of **D**. The indices of the surrounding elements are defined as the indices of **p** rounded upward and downward in each direction. Initially, there are four linear interpolations in the y direction followed by two linear interpolations in the x direction with the resulting coordinates and a last one in the z direction (a scheme can be seen in Figure 3.12).

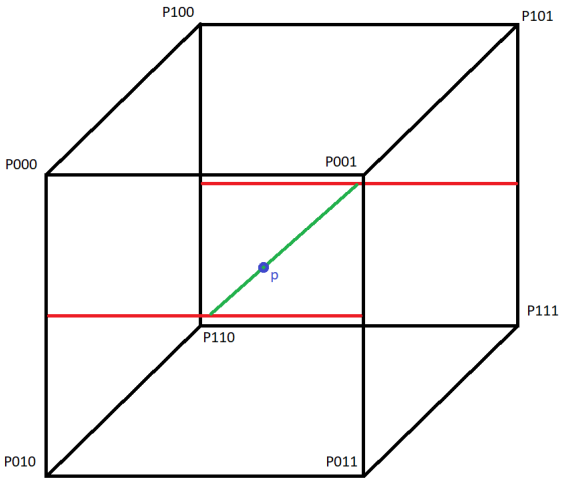


Figure 3.12: A scheme of the principle of trilinear interpolation. It consists of seven linear interpolations. First four interpolations in the y direction (results in the endpoints of red lines), second, two in the x direction and the last one in the z direction which ends in the trilinear interpolation between the eight corner points.

The weights for the interpolations can be extracted by the difference of the elements of **p**

with their downward adjusted versions (Equation 3.79).

$$\begin{aligned} d_x &= p_x - \lfloor p_x \rfloor \\ d_y &= p_y - \lfloor p_y \rfloor \\ d_z &= p_z - \lfloor p_z \rfloor \end{aligned} \tag{3.79}$$

The first four interpolations in y direction are shown in Equation 3.80.

$$\begin{aligned} P_{00} &= P_{000} \cdot (1 - d_y) + P_{010} \cdot d_y \\ P_{01} &= P_{001} \cdot (1 - d_y) + P_{011} \cdot d_y \\ P_{10} &= P_{100} \cdot (1 - d_y) + P_{110} \cdot d_y \\ P_{11} &= P_{101} \cdot (1 - d_y) + P_{111} \cdot d_y \end{aligned} \tag{3.80}$$

Next, the two interpolations in x direction are calculated as shown in Equation 3.81.

$$\begin{aligned} P_0 &= P_{00} \cdot (1 - d_x) + P_{01} \cdot d_x \\ P_1 &= P_{10} \cdot (1 - d_x) + P_{11} \cdot d_x \end{aligned} \tag{3.81}$$

The last interpolation is between the values of P_0 and P_1 (Equation 3.82)

$$P = P_0 \cdot (1 - d_z) + P_1 \cdot d_z \tag{3.82}$$

The result of this method is smoother without step artifacts and different to the nearest neighbor technique, multiple selections of the same points because of the rounding process are not possible. On the other hand, the trilinear interpolation needs more calculation time and the values are calculated, so they are not contained in the data.

Back to the Direct Volume Rendering (DVR) part, the start and endpoint of the ray are missing. It would be possible to take the data matrix $\mathbf{D}$ and sample in one chosen direction starting with the first index 0 and end in the last index m, n or l. For single volume visualizations which are axis-aligned (i.e., the direction of the rays are along a dimension of D), this would be an easy approach. If there are arbitrary directions of the rays with more than one dataset, there are more complex but also fast methods to solve this general case elegantly.

3.8.1 Ray Parameters by Intersections

The first one described in this thesis works with ray box intersections [Smi02]. In the visualization, a ray R is a straight line and the volume matrix corresponds to a cube or box in a Euclidean space. So it is possible to determine for each ray if there are intersections with the box representing the volume and to calculate the positions of the intersections if they exist. It is possible to get only one intersection on the corner then

the value is simply the corner value or this ray is threaded as no-hit. A ray without an endpoint can be described with the line equation (Equation 3.83):

$$R = \mathbf{o} + t \cdot \mathbf{dir}, \tag{3.83}$$

where $\mathbf{o}$ is the origin of the ray, t is the sampling position and $\mathbf{dir}$ is the normalized direction. The origin of the rays are normally on the viewing point of the visualization (i.e., the eye point). To not sample the empty space between the viewing point and the box, the startpoint for the sampling process has to be calculated. Let $B = (\mathbf{lbf}, \mathbf{rtb})$ be the definition of an Axis Aligned Bounding Box (AABB) with $\mathbf{lbf}$ is the left bottom front (minimum values of the box) and $\mathbf{rtb}$ the right top back corner (maximum values of the box). As the name implies, AABB are boxes that are aligned to the three main axes. In the next step, Oriented Bounding Box will be defined as a generalization for arbitrary rotated boxes. For the intersection task, the box is handled as a set of six planes and the calculation is being done separately for each of them. The positive effect of the axis-alignment is that the equations for the planes can be simplified to the respective corner values (e.g., the top plane is defined simply as the y element of $\mathbf{rtb}$).The following Equation 3.84 shows the six plane equations which are set equal to the ray and then solved for the unknown variable t.

$$\begin{aligned} t0x &= \frac{\mathbf{lbf}_x - \mathbf{o}_x}{\mathbf{dir}_x} \quad , \quad t1x = \frac{\mathbf{rtb}_x - \mathbf{o}_x}{\mathbf{dir}_x} \\ t0y &= \frac{\mathbf{lbf}_y - \mathbf{o}_y}{\mathbf{dir}_y} \quad , \quad t1y = \frac{\mathbf{rtb}_y - \mathbf{o}_y}{\mathbf{dir}_y} \\ t0z &= \frac{\mathbf{lbf}_z - \mathbf{o}_z}{\mathbf{dir}_z} \quad , \quad t1z = \frac{\mathbf{rtb}_z - \mathbf{o}_z}{\mathbf{dir}_z} \end{aligned} \tag{3.84}$$

Now there are six t values for plane intersections, but only two of them are the real intersection points with the box. The others are effects of the fact that the planes have no borders. Unless the ray and a plane are parallel, they will have an intersection. To determine the appropriate t values (let them be t_{in} and t_{out}) some comparisons are needed. In the beginning, the two values are set to the results of $t0x$ and $t1x$ where t_{in} contains the minimum value of them. Next, the first comparison step with the results of $t0y$ and $t1y$ has to be done. There are four possible cases:

- $t_{in} > Max(t0y, t1y) \Rightarrow$ no intersection
- $Min(t0y, t1y) > t_{out} \Rightarrow$ no intersection
- $Min(t0y, t1y) > t_{in} \Rightarrow t_{in} = Min(t0y, t1y)$
- $Max(t0y, t1y) < t_{in} \Rightarrow t_{out} = Max(t0y, t1y)$

Next, the comparisons with $t0z$ and $t1z$ have to be done. The four cases are the same as for $t0y$ and $t1y$ only $t0z$ and $t1z$ change to $t0y$ and $t1y$. In the end, t_{in} and t_{out} hold the target values. In case of a negative intersection point, the ray origin is inside the box and

the startpoint for the sampling can be set on the ray origin. If the box is not axis-aligned, like a OBB, there is a rotation matrix which rotates the object from the axis-aligned 'object space' in a 'world space'. If this rotation matrix is inverted and applied to the ray, it is possible to use the AABB intersection test to get the intersections with the box.

Now we have the intersections of a ray with the box representing the volume. The last values missing are the origin and the direction of the ray. If the calculation is being done on the GPU, every fragment/pixel shader corresponds to a ray and the direction is the (sometimes perspective distorted) viewing direction of the camera. Formal it is also possible to define a camera and calculate the ray direction. Let $Cam = (\mathbf{p}, h, w, fov_x)$ be a camera, where $\mathbf{p}$ is the position in Euclidean space, h and w are the height and the width of the screen and fov_x is the field of view in x direction. fov_y can be calculated with $\frac{h}{w} \cdot fov_x$. h and w are integers greater than zero and the field of views are in the range $(0, \pi)$. The field of view defines the degree of perspective distortion. Let x_p an integer between 0 and w, and y_p between 0 and h be a point on the screen where a ray should be traced. Equation 3.85 describes the formula which can calculate the position (x, y, z) of this point (x_p, y_p) respective to the camera parameters in 3D space.

$$\begin{aligned} x &= \frac{2 \cdot x_p - w}{w} \cdot tan(fov_x) \\ y &= \frac{2 \cdot y_p - h}{h} \cdot tan(\frac{h}{w} \cdot fov_x) \\ z &= -1 \end{aligned} \tag{3.85}$$

The z coordinate is the position of the viewing plane. The origin of the ray can be set to $(0, 0, 0)$ and the corresponding direction is the normalized difference of the origin and the result of Equation 3.85. A scheme of this calculation is shown in Figure 3.13.

3.8.2 Ray Parameters by Textures

The second approach to get the necessary components for ray casting uses the advantages of shaders and buffers [KW03]. Like the method explained before, this technique uses a box which contains the volume, but this time it is not only described by its minimum and maximum corner. The box is defined as a set of triangles which are two lists: a corner list and an index list which describes which corners forms a triangle. In short, this algorithm needs three render steps. Two to get the startpoint and endpoint for the sampling, and one render step for the actual sampling of the volume. In the first step, the front faces are rendered into a texture which means all occluded faces are not visible anymore. Next, the back faces, which are the sides of the faces which showing inward the box, are rendered into another texture. In the last step, the sampling of the volume is done. Every fragment/pixel shader call represents a ray. The startpoint is the value stored the front-face texture. The endpoint is the value in the back-face texture and the direction is the difference between these two textures.

Now, this three rendering steps are described in detail. This technique needs some proxy geometry namely a box and a plane. The plane is for the third rendering step and

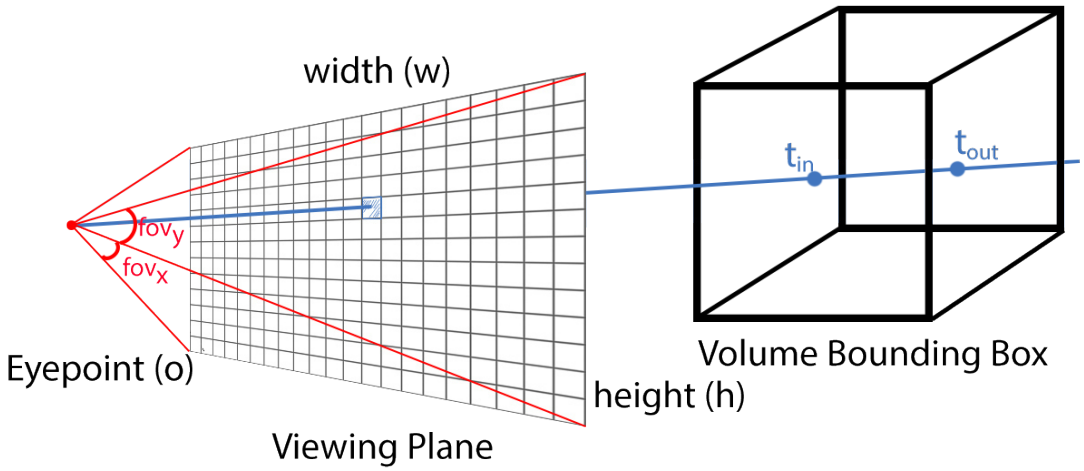


Figure 3.13: The principle of volume rendering with ray casting. A ray is emitted from the origin through all pixels of the viewing plane. t_{in} and t_{out} are the intersections points with the box of the volume and represents the start and the end point of the volume sampling.

contains the image that should be displayed. It is possible to have 3D texture coordinates assigned on the edges of the box as additional information but they can also be calculated from the 3D positions of the corners (described later). Since the plane is for displaying the final result on the screen, it would be best if the corner points of it have the 3D position $(-1,-1,0)$, $(-1,1,0)$, $(1,1,0)$ and $(-1,1,0)$. The box is the bounding object containing the volume information and can also be used for transformations (translation, rotation, scaling, ...). The transformations will also affect the volume. As every object in a scene, the box has a 4×4 transformation matrix (called the model matrix) which contains the information how it is oriented and placed in 3D space. This matrix fulfills the task of a transformation from the objects space, where they are defined and created, and the world space where multiple objects can be composed to a scene. If the resolution of the volume is not equal in each dimension, a scaling matrix can be included into the model matrix. This way, the box can be adapted to the ratios of the resolution of the volume and the volume gets not distorted in the visualization. The scaling factors can be calculated by dividing every dimension of the resolution with the maximum dimension. The matrix for the scaling contains only zeros except for the diagonal elements. These elements contain the scaling factors and a one in the right lower corner. Equation 3.86 shows a scaling matrix.

$$\mathbf{S} = \begin{pmatrix} \frac{n}{Max(n,m,l)} & 0 & 0 & 0 \\ 0 & \frac{m}{Max(n,m,l)} & 0 & 0 \\ 0 & 0 & \frac{l}{Max(n,m,l)} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.86}$$

Multiplying the model matrix of the box with this scaling matrix results in the new matrix which takes the dimension ratio into account. Generally, the model matrix is first multiplied with the scaling matrix, then with a rotation matrix and in the end with the translation matrix if they are available. If the geometry is set up, the three render steps are applied to them. In the first step, the front faces of this box are rendered, which is simply a normal render step. This rendering is done to get the startpoints for each ray in the third rendering step. The important part in the first step is, that the 3D texture coordinates assigned to the corner vertices of the box are used as 'color' and that the render target is a texture (namely front face texture) and not the screen. The texture coordinates can also be calculated by adding one to the vertex position (before the multiplication with the transformation matrix) and a division by two. This works only if the box is in the range of $[(-1,-1,-1,1),(1,1,1,1)]$. If the box is in the range of $[(0,0,0,1),(1,1,1,1)]$ the texture coordinates are equal to the vertex positions. It is important to map the maximum vertex coordinate to one and the minimum one to zero. Therefore this mapping is done in the vertex shader, the Graphics Processing Unit (GPU) does all of the interpolation work. On the fragment/pixel shader side, these texture coordinates are written in the output color (with an additional one as alpha value). Normally, back face culling is activated during the rendering process with some exceptions like for transparent objects. This means, that all faces which normally show in the same direction than the viewing directions are not drawn (the order of the verices of the triangles can also be used, to detect such faces [Ebe06]). In the second rendering step, the endpoints of the rays are determined. For this, the back faces of the box are from interest. If only the back faces should be rendered, the front face culling has to be activated. Now the box is drawn in the same way as for the front face texture without any further changes. The created texture is called the back face texture. With these two textures (front face and back face), all information needed is available. In the third render pass, the culling is returned to back face culling and the plane is rendered on the screen to display the volume. The texture coordinates for the plane can also be calculated like the coordinates of the box, but this time, the z coordinates are dropped because the plane needs only 2D values. In the fragment/pixel shader, the front and back face textures are sampled with the texture coordinate of the plane to get the start and the endpoint of the current ray. The direction corresponds directly to the difference between these two vectors. With the Equations 3.76, 3.77 and 3.78 the ray can be sampled to determine the right output color.

3.8.3 Ray Sampling Methods

To create a color for the display on screen the sampled values on the ray has to be used. There are a high amount of techniques which can be used. Some methods on how the resulting color can be composed with the sampled values are described now. Starting from the simple ones to the more complicated ones. Every technique has its benefits for specific tasks which will also be listed.

A common method is the Maximum Intensity Projection (MIP) [SGS95]. The result color is based on the maximum value sampled along the ray. The value is often used to create a gray value (the red, green and blue value are set to the max value), but it is also possible to assign a color lookup table and set the color from there. The resulting visualization shows the densest parts in the volume (like bones, shells, ...). An advantage of this method is that the colors correspond directly from the data without any changes. That is why it is often used in medical visualizations. Another method, which uses the data values directly for the colors is the first hit method. The first data sample found on the ray is used (samples of value zero are skipped). This results in a surface image of the volume. Mostly the background of the scans contains a certain degree of noise and this technique is extended and the first hit after a predefined threshold is used. If an x-ray like visualization is desired, the mean value of the sample values is used. If the only values directly from the data should be used, the nearest value to the mean value can be used. The median value is also a good choice, because it is more robust against outliers and can be used directly. For an image of a specific view aligned slice of the data, every ray can be sampled once, on the same distance to the camera.

All of these methods, except the average one, use only one value of the ray and discard the other information. The next described method is alpha compositing which aggregates the values weighted with their alpha values. This can be done with the help of the Porter Duff Algorithm [PD84]. Respective to compositing there are two processing orders: front to back or back to front. Let $\mathbf{col_{in}}$ be the input-color, $\mathbf{col}$ the actual sampled blending color and $\mathbf{col_{out}}$ the composed out-color. Let also be α_{in}, α, and α_{out} their respective alpha values. The calculation of the color starting from the back face coordinate is shown in Equation 3.87:

$$\mathbf{col_{out}} = (1 - \alpha) \cdot \mathbf{col_{in}} + \alpha \cdot \mathbf{col}. \tag{3.87}$$

For the front to back approach Equations 3.88 are used.

$$\begin{aligned} \mathbf{col_{out}} &= \mathbf{col_{in}} + (1 - \alpha_{in}) \cdot \alpha \cdot \mathbf{col} \\ \alpha_{out} &= \alpha_{in} + (1 - \alpha_{in}) \cdot \alpha \end{aligned} \tag{3.88}$$

It can be seen that the back to the front calculation is less complex which is more obvious if the colors are premultiplied with their alpha values. On the positive side of the front to back approach is the possibility of early ray termination. If the alpha value is close to one, all samples behind the sample that changed the alpha value to one will not affect the color and can be skipped to speed up the runtime.

An advantage of the alpha compositing is to show different tissues in different colors. In this case, it is possible to visually segment the volume and show the surrounding context at the same time. Transfer functions are used to translate values into colors. There are many possibilities to create such functions [LKG+16]. The simplest one creates a color directly from density values which are in the range 0 and 1. For example the color: (sampled value, sampled value, sampled value, alpha). So each sampled value

corresponds to a gray color. The alpha parameter controls how deep the ray can penetrate the volume and can be a fixed value or depending on the density value. The described simple transfer function maps the values from white to black. Another easy transfer function is a linear interpolation between colors with the density value as weight. These transfer functions map the sampled values only between two colors. Sometimes it is better to use different colors for different density areas. The interpolation approach can be extended by using four functions, one for each color channel plus the alpha channel. If a texture which contains a color gradient is available, it can also be used with the density as sampling coordinate. Finding one fixed function for arbitrary volume data is not possible. Finding a suitable transfer function calculated from the data is a big challenge in computer graphics [LKG+16]. If it is possible, it is a good idea to integrate the knowledge of the domain expert user and let them change the parameters. In the case of the transfer function, this could be a widget which contains four graphs. The axes are horizontally the sampled density values and vertically the influence of each channel. The user can change the control points to influence the colors and the alpha values. To determine the resulting color of the density the values of the nearest control points are used for the interpolation. To help the user, for example, a histogram of the density values can be visualized in the background of the graphs.

CHAPTER 4

State of the Art

In this chapter scientific works similar to the tasks of this thesis are introduced and the differences to the topic of this thesis are worked out. To cover all requirements of the task, on the one side, works from the seed classification domain (Section 4.1) and, on the other side, Visual Analytics systems to support the classification of 3D and 2D data are discussed (Section 4.2).

4.1 Seed Classification

The computational classification of archeological seeds is a rather undeveloped field and restricted to very specific plant groups. The work with the most similar topic to this thesis is the classification of wild and domesticated grape seeds [UOG+16]. This group of plant is selected because of their importance since the beginning of agriculture in the Mediterranean area. The authors use modern and archeological seeds and create a database by scanning them with a flatbed scanner to obtain 2D images. For these images, they calculate different shape features for the classification. Additionally, they calculate Elliptic Fourier Descriptors for the contour shape. As classifier they use Linear Discriminant Analysis (LDA). The goal of the authors was to use as few features as possible to get a good model for the classification. For this, they define three criteria to determine which features are added into the model: the Tolerance, the F-to-enter statistic and the F-to-remove statistic. Initially, there is no feature in the model and the features are added one after another if they fulfill one of these criteria. In case of F-to-remove, all features are in the model initially. The Tolerance measures the colinearity of a feature (i.e., how related the classes are to each other). A new feature which is not in the model is treated like a dependent variable and regressed to the other independent variables. The Tolerance is defined as $1 - R^2$, where R^2 is the coefficient of determination. R^2 measures how good the regression fits the underlying data. If $R^2 = 1$ all points are on the same line. F-to-enter is a measurement of how the model changes if the feature is added to the

model. F-to-remove measures how the model changes if the feature is part of the model and then removed. The calculation of this test (F) can be seen in Equation 4.1 [Pas02]:

$$F = \frac{SSR_m - SSR_{m-1}}{\frac{SSR_m}{sc-m-1}}, \tag{4.1}$$

where SSR_q is the squared sum of residuals of all features in the model plus the feature to test. SSR_{q-1} is similar but only containing the features in the model without the new feature and sc is the number of samples. Since the model is created step by step, m is also the number of features in the model. The residuals are calculated by subtracting the value given of the regression from the actual value. F-to-remove works similarly, but starts with all features in the model and calculates F by removing one feature by another. A threshold for the F-to-enter value is defined to determine the features that should be considered in the final prediction model. The authors of the paper use the value of 3.84 as a threshold for the F-to-enter value to determine which features should be in the model. All features below this value are not considered in the final prediction model. Since in this thesis, the data is a set of 3D volumes and every calculation is done in three-dimensional space. There are more different groups of plants to differentiate with classes which only contain a very few number of seeds. It should also be possible to visually check the results of the learner which was not possible in this explained paper. The domain expert should be more integrated into the decision process.

The classification of fresh seeds is more common than the classification of carbonized ones. An approach for Rubber Seeds is shown in the paper by Hashim et al. [HOAJ+10]. They also used a camera setup to gain 2D images of the seeds. With edge detection they get a gray scale image of the data. Additionally, morphological operators are used to get the seed fully filled and small holes are closed to create an image which only contains the contour. From these images, they calculate the area and the perimeter. They also calculate 36 contour point radii, which are the distances of a point on the contour to the centroid. These features are used to train an Artificial Neural Network (ANN) classifier. In a second approach, they reduced the dimensionality of the feature space with the help of PCA and train the ANN with this reduced features. ANN is the generalized name for all neural networks. Also CNN classifiers are a subclass of them. The difference between ANN and CNN is that an ANN has, in general, no convolution layers to detect local features. They get a set of feature vectors and learn the best regression parameters for classification [BH00]. Since they use a non linear function during the learning process, even complex regression lines which are not only straight lines or curves can be accomplished. In the paper, they use a feed forward approach, which means the learning is done from the input to the output (opposite to the back propagation described in Section 3.7 (CNN)). As the best fitting method of the regression, the Levenberg–Marquardt algorithm is used [Mar63]. This algorithm is also called least squares curve fitting. The Jacobi matrix used in this algorithm contains the derivatives of the network errors to minimize them.

A general recognition system of different seeds (called PSIRS) is described by Lurstwut et

al. [LP11]. The data also consists of 2D images obtained with a camera. Like the paper of Hashim et al. [HOAJ+10], they create a binary image in a preprocessing step, by thresholding this images and applying morphological operators to close all holes within the seeds. As features, they use shape measurements (like the boundary pixel count and the roundness). A feature called 'ripples' is also used which is the number of pixels a seed differs from an average seed of the same class. For this, the binary image is subtracted by an image with an average boundary of the seeds of this class. Lines thinner than 10 pixels are removed and the remaining components are the ripples. They count them and calculate the number of pixels within each ripple. Another technique to get features used is to split the image in an upper and a lower area with equal size and calculate the characteristics of the half images. They calculate the ratio between the seed area and the overall area for both components. The seed area is the sum of all pixels within the seed which are in fact the ones in the binary image. Since the original images are captured by a camera, the colors can also be features. They use the average RGB color and the average L*a*b* (of the L*a*b* color space [MT09]) color to store the color information of the seeds. Finally, some texture features are calculated to gain extra information from the surface of the seeds. More precisely, the energy (Equation 3.31), the entropy (Equation 3.30), the contrast (Equation 3.26), the homogeneity (Equation 3.28), and the correlation (Equation 3.32) features are taken into account. To classify new samples, the Euclidean distance of all m features is considered as shown in Equation 4.2:

$$ED = \sqrt{\sum_{i=0}^{m-1} (x_i - p_{new_i})^2}, \tag{4.2}$$

where x_i is the value of feature i of the samples of the training dataset and p_{new_i} the values of the new sample.

If the classification challenge is generalized to CT data classification, most of the recent works make use of Convolutional Neural Networks (Convolutional Neural Network (CNN)s) [MMH+17] [GHT17] [PDM19]. These networks (specified in Section 3.7) learn features directly from the slice or volume data. This is why there is no need to select the best features in the data, because they are found automatically. The downside of CNNs is that usually a big amount of labeled training data is needed to get proper results. If the classes have only a few samples, tree based classifiers are also a powerful option, but with the need of precalculated features [YLY+20]. In our case, description of the seeds, that can be exported to a file, should also be available. Because of this task, many features used for the descriptions are available and can directly be used as learning features for a non CNN classifier. Some classes also only have up to four seeds, which is normally too few for the CNN approach.

An approach which combines feature extraction and an ANN is introduced by Lakshmanaprabu et al. [LMS+19]. They calculated a set of histogram, texture and wavelet-based features to train a classifier which can detect lung cancer in CT scans with a very high accuracy of 96.2%. Because, most of these features are not well understandable for

humans, they can not be directly used for the purpose of this thesis. The goal of the paper was also a fully automatic approach, which provides no interaction with a user during the decision making.

4.2 Visual Analytics Systems

On the visual analytics side, there are user driven classification approaches without machine learning. An example is the iVisClassifier [CLKP10]. This software uses the LDA (see Section 3.7) method and offers the user interaction techniques to explore a loaded feature dataset. The user can interactively reduce the number of dimensions to get a plot that allows good classifications. A Screenshot of the user interface is shown in Figure 4.1. Since the domain expert likes to have class suggestions because of the amount of unknown data and the lack of time, a machine learning part has to be integrated. The shown datapoints in the plots are the LDA components and not the features themselves which should be explorable. A LDA was also used in the software of this thesis, but as additional view to the calculated shape features and not as stand alone.

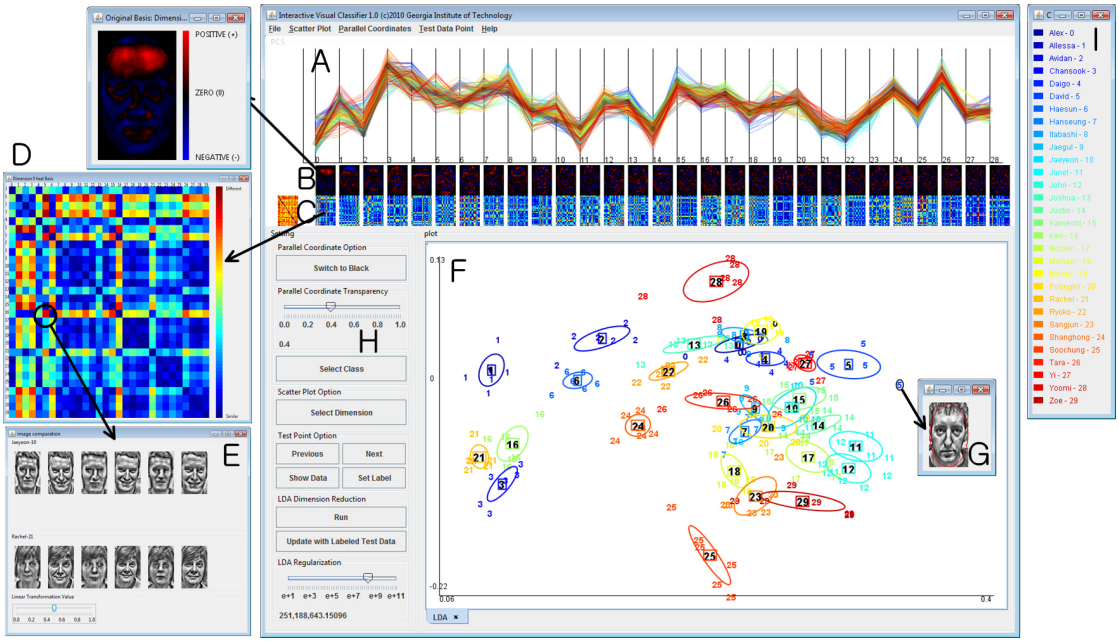


Figure 4.1: User interface from the iVisClassifier [CLKP10] used to manually classify data with the help of a Linear Discriminant Analysis (LDA).

In the paper of Paiva et al. [PSPM14], a method is presented which improves the machine learner with the help of the knowledge of a user. The user can select the most important samples and change their labels during the model creation process to get better predictions. New samples are added to the model incrementally by controlling

and correcting the suggested labels by the user. This can be done with several different machine learning methods. Due to the high number of classes this is also a time consuming task for the domain expert and it is very unclear which of the seeds would be more relevant for the classifier than others, the model improvement step will be as simplified as possible.

A Visual Analytics method to explore the result of a probabilistic classifier is presented in the paper of Alsallakh et al. [AHH+14]. The aim of this work is to create a better overview of the incorrectly classified samples depending on their probabilities given by the classifier. As central technique they introduced the confusion wheel view. This view is an improved version of the often used matrix of confusion. The different classes are placed around a wheel, instead of row and columns in a matrix. Within the wheel, arcs are used to connect classes with samples that are misclassified to each other. To visualize the number of samples, the thickness of the start and the end of the arcs is used. A broader arc diameter means more confused predictions. Additionally, the rates of FP (false positive), FN (false negative), TP (true positive), and TN (true negative) classifications are shown as bar charts in the class sectors. A scheme of this plot is shown in Figure 4.2. Since the authors work with probabilistic classifiers, multiple bar charts

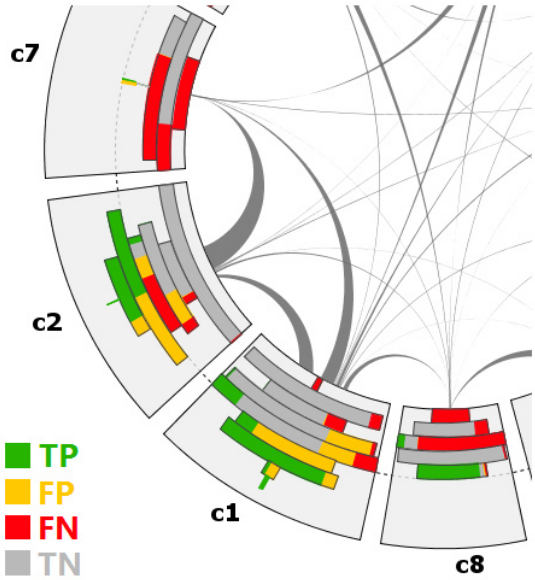


Figure 4.2: A scheme of the confusion wheel view [AHH+14]. The classes ($c1$, $c2$, ...) are arranged around a wheel. The arcs visualize samples that are confused to the other classes. The number of confused samples is encoded by the width of the endpoints of the arcs. The bar charts contains the distribution of the four sets FP (false positives), FN (false negatives), TP (true positives) and TN (true negatives) on a specific probability output of the classifier.

are shown for different probability values. This bar charts are set on rings around the wheel. For example on the 50% ring, the distribution of the FP, FN, TP and, TN marked samples are shown where the classifier has assigned the samples with a probability of 50% to a specific class. This tool has also other useful views, like a feature importance plot as a boxplot or comparison methods for different classifier outputs in the confusion wheel and the boxplots to check if the new classifier or the changed parameters of the actual classifier is an improvement or not. As already mentioned, the improvement part of the model in this thesis should be only very simple (for example by deleting outliers and other problematical samples), the refinement of the learner model is not necessary. Whereas the confusion wheel is a interesting visualization, the interesting part in this work lies in the positions of the incorrectly classified samples according to specific features. This should be visible directly while exploring the data.

CHAPTER 5

SEEd

SEEd is the name of the software resulting from this work. In the following, the user tasks are listed on a high level followed by the more detailed descriptions of the low level tasks:

- A visual and descriptive overview of a high number of different seed species with human understandable descriptors.
- Visually separate the seeds species which are very similar to each other.
- Offer species suggestions for unknown seed and support the domain expert during the classification process.

The first low level user task is to load the existing dataset. The ancient seeds are stored in the DICOM format and the lab-primed fresh seeds are stored in the TIFF file format. It should also possible to load whole datasets at once with mixed filetypes. Additionally, there are several internally used data types, which should also be loadable. To fulfill this requirement and to easily add more supported data types in the future, a separate data loading step is integrated.

Another low level user task is the visual inspection of the loaded data like the surface or the inner structures. This implies the use of different DVR techniques: on the one side of the object within a file directly and on the other side of the segmented and separated single seeds within a file (if available). The segmentation is also necessary for all following tasks, because the features have to be extracted per seed and not per file. Since the segmentation is a non trivial task, the extracted seeds should be saved for a fast loading. In the DVR plot, zooming and rotating is possible. The functionality of this low level user task is summerized in the preprocessing step.

The next user task is to get a descriptor (i.e., a set of different feature values) of every seed and export them to a file for storage purposes and for use in other software. Different shape features are extracted to get a good description of the seeds. As filetype, Comma Separated Values (CSV) is selected, because it is a common type for tabular data. It should also be possible to check, if the segmentation and the feature calculations are correct.

The main task of the domain expert is the classification of unknown seeds. It would be best, if the new seeds belong to species which are already in the data base, but it is also possible to have seeds which are completely unknown. Reasons for a seed beeing unknown are for example: not scanned, extincted over time, changed during evolutionary processes. In this case, it is important to have suggestions which species comes closest to the new one and which biological genus fits the best. To support the decision process, the feature distributions are visualized. Because it is easy to understand and navigate, a scatter plot is used as main visualization. A preselection of the two axes with a good separation of the classes is offered, to show the domain expert directly a helpful plot. Because of the number of datapoints, zooming and navigation is possible. Other supporting visualizations are offered optionally to emphasize the boundary and the area of high densities within the selected axes of each class. Also selection methods are available to show specific seeds in the DVR widget. To remove incorrectly segmented seeds or outliers, which only distort the classes of the data, selected seeds can be deleted. If species are not necessary for a specific classification, whole classes (i.e., biological species) can be hidden for a better view of the useful species. There are much reasons to hide species. One would be if outliers are searched and a species do not have any, or species that can be definitely excluded of the possible classes during the classification process. For fast loading, an internal filetype is used to store the feature list. In the pipeline, all this functions are summarized in the feature extraction step.

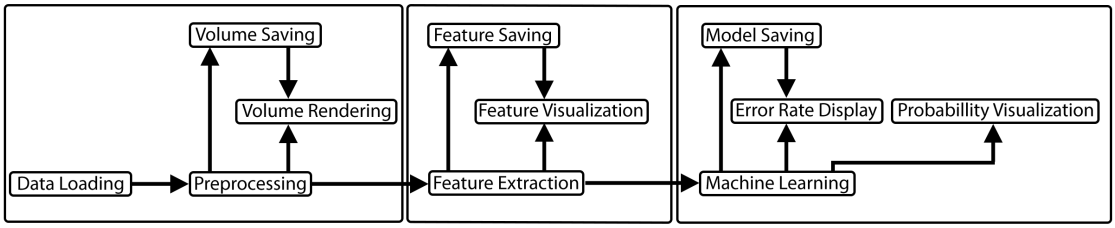


Figure 5.1: The SEEd software pipeline. Details for the three sub-pipelines are given in the respective sections. For a better overview, the arrows between the rendering/visualization steps (Volume Rendering, Feature Visualization and Probability Visualization) are not drawn, but all of them interacts with each other. The arrow represents a data transfer and the boxes are procedures.

Since there is a high number of unknown seeds to classify, a machine learning step is also integrated. This step exists to offer the domain expert a list of species with

probabilities the unknown seeds could belong to, to sort out biological species that are irrelevant for the classification in the best way possible. The domain expert has to know how accurate the learner is. During the learning process, a cross validation is performed. The accuracy value is visualized in the UI, well visible with a color code to show visually how good the classifier is. The incorrectly classified seeds are also visible in the scatter plot to visualize the problematic regions within the plot where many missclassifications occur. In Figure 5.1 a high-level overview of the processing pipeline is outlined. More details of each of the three sub-pipelines (preprocessing, feature extraction and machine learning) are given in the next sections. For a better overview, the interaction arrows between all rendering/visualization steps (the middle row in the figure) are not drawn. As, the loading part is too short for a section of its own, it is put into the preprocessing section.

5.1 Preprocessing

The data contains carbonized seeds from different plant species, mainly food grains like peas or lentils. There are three origins of the data samples. The samples include ancient seeds encountered in the Fertile Crescent up to 10.000 years old, fresh seeds synthetically carbonized in a lab and fresh uncarbonized ones. All of them are embedded in polymers to get a better background contrast in the subsequent 3D scans. These are produced by the domain expert with an industrial Computed Tomography (CT) scanner in the Biodiversity Center of the University of Vienna. There are compact CT devices available and through the storage properties of the samples the scanners can be used to get high-quality volumetric data. The functionality of the scanners is described in Section 3.1. The filesizes and types are described in Section 3.2.

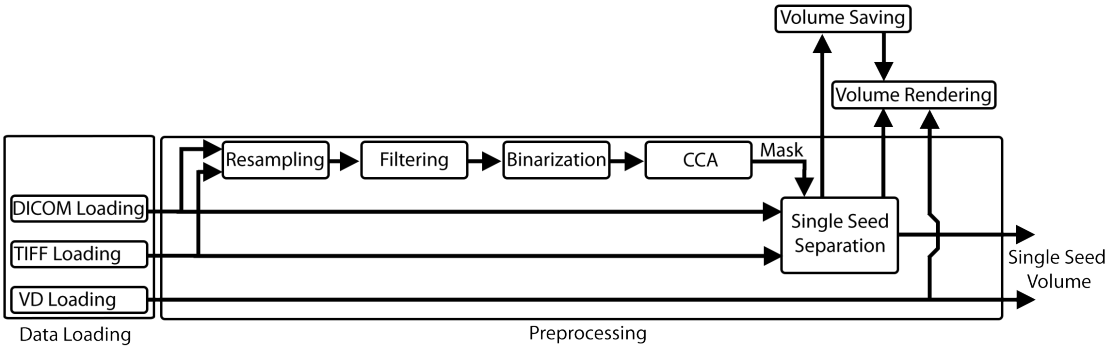


Figure 5.2: A scheme of the preprocessing pipeline including the loading step of the SEEd software. The VD (Volumetric Data) datatype is an simple self-made file format for single seed volumes to speed up subsequent loadings. DICOM and TIFF may contain multiple seeds per file so they have to be separated from each other.

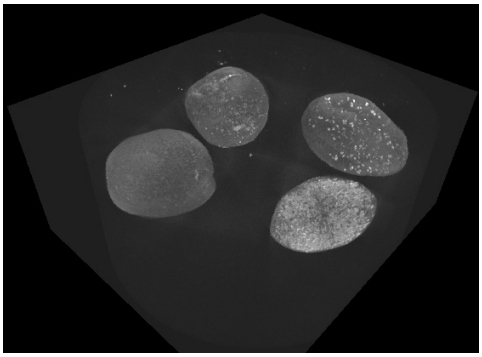
Details of the preprocessing step of the pipeline are shown in Figure 5.2. Initially,

the data files have to be loaded. In case of adding new test samples to a dataset and classification model, the seed files have to be in directories named by the seed species composed in a single root directory. To speed up the process, it is possible to set a property of the software that the files contain already single seeds. In this case, the segmentation and separation processes are not necessary.

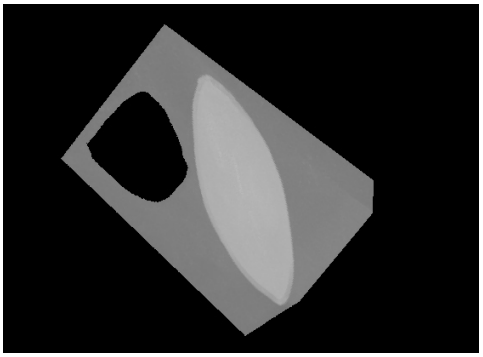
During the loading process, the values of the dimensions and the density information are stored into an internal data structure. The ordering of the raw data is in $m \rightarrow n \rightarrow l$ direction of the volume. In case of the DICOM format, the slices are stored in several files with an additional file named DICOM, which contains the information of the files belonging to the volume. Theoretically, it is possible to store multiple volumes in one DICOM directory, but practically this is not the case in our dataset. The file paths of the slices are extracted from the DICOM file, and the slices are loaded one by one. In the TIFF files, the slices are stored in a single file per volume. The VD (Volumetric Data) files are structured like the internal used data structure containing the dimensions of the volume resolution and the density values in the ordering needed ($m \rightarrow n \rightarrow l$) to speed up the loading process. It is possible to load a whole directory which contains multiple species directories with multiple seed files. There is also a loading option for single seed files or single slices of the DICOM file. Single seeds can be loaded with feature calculation in the next step or only for displaying. After the loading process, the output is an equally data structure for all file formats. This way it is easily possible to add loaders for other file formats without any changes of the other pipeline steps.

Since the loaded VD file data containing only a single file, a segmentation is not nessesary and the preprocessing step is skipped for files of this filetype. This is also the case if the single seed option per file is used. Otherwise, it has to be checked if there are several seeds within the volume. Additionally, masks to extract them have to be created. This has to be done, because the features will be calculated per seed. There are also files that contain parts of other seeds because of slicing errors or other reasons. These have to be detected because they should not be considered in future steps. Some of the files contains scanning artifacts as well. Images of these cases are shown in Figure 5.3. For the segmentation masks, the volume is first resampled to $\frac{1}{12}$ of its resolution (see also Section 19). To get this number, several down sampling rates are tested and $\frac{1}{12}$ is the rate where are no segmentation errors accrue and the runtime decrease significantly. This step is done to reduce the calculation time of the next steps without loosing structures after the separation step. In the filtering step, a median filter (Section 3.3.2) is used to reduce the background noise. Other filtering techniques are tested as well but without sizable improvements. For the binarization (Section 3.3.2) of the image, the Otsu threshold (Section 3.3.1) is used. Because of different noise intensities, it is not possible to use the same threshold for all images and this adaptive method is selected.

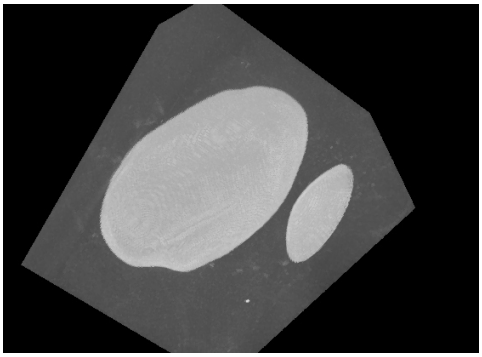
Now, a CCA (Section 3.3.2) can be used to detect all connected components. For this, a two pass method is used. This method works best on binary images. Since



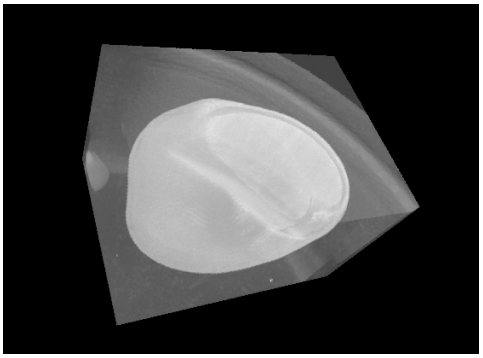
(a) A normal DICOM file. Multiple seeds in a round storage box.



(b) Big holes into the files generates a third pin into histograms.



(c) Parts of other seed visible. This fractions of seeds have not to be considered.



(d) Artifacts of the storage boxes can also distort the histogram of the data.

Figure 5.3: Images of raw data samples which contains special cases which have to be considered during the loading process.

there will be many components found, they have to be filtered. One method used is the size of the components. If the size is smaller than half of the biggest object, it is not considered a seed. Since the seeds of a file are always from the same species, they will have roughly the same size. It is also possible that there are components within a bigger component because of the binarization and the nature of the seeds having a gap between the shell and the inner structures. In this case, the distance between the centroids of the components is checked. For this, the bounding radius for each component is calculated. If the distance is smaller than the bounding radius of another component, the smaller one is discarded. For all components that are not filtered out are the bounding boxes calculated. These are the smallest boxes enclosing the components. To get the separated seeds in its original resolution the boxes have to be back transformed with the inverse shrinking factor, which is 12 in case of this work, of the resampling step. A small bias of 15 voxels is added, because of possible missing structures during the shrinking process, for the use on the original data. The boxes are now used to extract the seeds. The parts in the bounding boxes are extracted and stored in separate internal used seed data

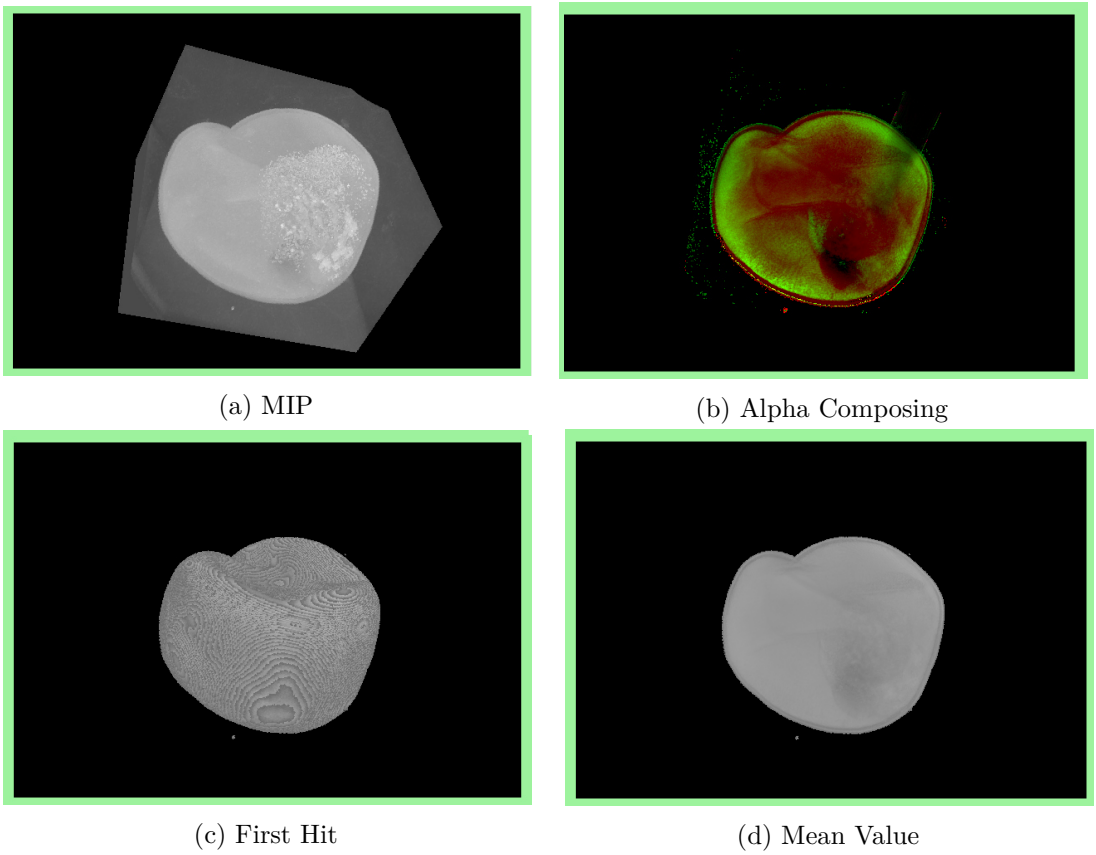
structures. If the bounding box which are enhanced by 15 voxels extends the boundary of the volume, there is a high probability that it is only contains a fractional seed on the edges of the volume and is also deleted. If desired, the newly created volumes can be saved automatically into separate VD files for fast loading and to avoid the separation step at every loading. A target directory can be selected and during the saving process, new sub directories for each species are created. In Algorithm 5.1, this segmentation and separation procedure is shown in pseudocode.

Algorithm 5.1: Check for Subvolumes

Input: A volume v containing the resolution as int values and a float list containing the density values.

Output: A list of volumes vl which stores the resolution, the density values as a float list, the path and a number of the subvolume.

```
1  create list vl;
2  resampled volume ← resample(v);
3  filtered volume ← medianFilter(resampled volume);
4  binary volume ← otsuThreshold(filtered volume);
5  connected components ← CCA(binary volume);
6  sort connected components in descending order respective to their
     NumberOfPixels;
7  for comp ∈ connected components do
8      if comp.NumberOfPixels > Max(comp.NumberOfPixels)/2 then
9          calculate the boundingsphereradius of comp;
10         if comp.centroid outside bounding radius of other components then
11             bounding box ← comp.boundingBox;
12             backtransform only the edges of the bounding box with inverse
                 resample rate;
13             add a bias to the bounding box;
14             if bounding box is entirely within the dimensions of v then
15                 newVolume ← extract voxels within the bounding box area from v;
16                 list vl.append(newVolume);
17             else
18                 comp is not a seed and is ignored;
19             end
20         else
21             comp is not a seed and is ignored;
22         end
23     else
24         comp is not a seed and is ignored;
25     end
26 end
27 return list vl;
```



(a) MIP (b) Alpha Composing

(c) First Hit (d) Mean Value

Figure 5.4: Images from the four DVR methods used in the SEEd software. To control the transfer function of the Alpha Compositing, an additional widget is available.

The created volumes can also be directly visualized via DVR (Section 3.8). This option is employing the domain expert the option to inspect specific files in their data without the whole processing pipeline. Possible rendering techniques are: MIP, alpha compositing, first hit and mean value (shown in Figure 5.4). For the alpha compositing technique, a transfer function widget is shown in which the user can modify the control points of the graphs for each color channel plus the alpha values. A histogram of the density values of the visualized seed is shown in the background of the transfer function widget to support the user to find the best functions. MIP is selected because it shows the densest parts of the seeds. With alpha compositing, the domain expert controls what is visible in the plot. The first hit method is available to inspect the surface of the seeds and the Mean Value shows a x-ray like version of the seed. An Otsu threshold is used to filter out background voxels. Since the threshold does not work with all seeds, the MIP version is normally rendered to have a view which always shows the data appropriately. Since the threshold for the DVR is only used for visual purposes, it does not affect the segmentation or other

steps. It should only blend out the background noise for the user.

The user can rotate the DVR visualization and zoom in/out. If multiple seeds are shown, the rotation is linked between the rendering widgets, so the same areas can inspected in all of the visualized seeds. The number of rendering widgets depends on the number of different species and can be calculated by $2 \cdot \text{number of species} + 1$. Two times, because every species gets a widget for the fresh samples and one for the carbonized samples. The extra DVR widget is for selections. An arc ball rotation is implemented for this user interactions. Some rotation methods have the problem that the angle getting inverted if the object was rotated over a certain degree (mostly $180^{/circ}$) beforehand. The arc ball technique pretends a virtual sphere around the object. After a mouse button is pressed and moved, the mouse position is transformed on the sphere and the rotation axes and angles are calculated from this transformed point. The zooming operation is done similar. At the beginning, the mouse position is projected on the sphere to determine the direction of the zooming and then the viewing point is translated in the direction of this point on the sphere. This way it is possible to zoom on specific positions on the volume. For this, the actual mouse position, is transformed into world coordinates. The z coordinate of the transformed mouse position is set on the volumes z coordinate, which is zero. The viewing point was set to -1. This coordinate is the endpoint of the trajectory starting by the actual position of the camera. A translation operation is then applied on the view matrix which depends on this trajectory towards or away from the volume according to the mouse wheel input.

Algorithm 5.2: Calculate Color

Input: An float index between zero and one, and a max number of colors needed.
Output: A Color.

```
1 h = index * (360/colorCount);
2 s = 200 # because of personal preference;
3 l = 200 # because of personal preference;
4 return Color(h, l, s);
```

For the linked rotation, the same rotation matrix is assigned to all widgets and premultiplied with the model matrices of the boxes representing the seed volumes (details in Section 3.8) before rendering. The boxes are also transformed with a matrix to fit the aspect ratio of the resolution of the seed volume. A zoom matrix is generated which has $\frac{n}{Max(n,m,l)}$, $\frac{m}{Max(n,m,l)}$, $\frac{l}{Max(n,m,l)}$ as the first three diagonal entries. This matrix is premultiplied with the model matrix of the volume to prevent distortions created from false aspect rations. Another premultiplication is done with the rotation matrix created from the registration to get an equal rotation to all seed volumes. This registration is done with a PCA. The PCA, done in one of the subsequent steps (Section 5.2), calculates the principal axes of the volumes. A rotation between these axes and the coordinate axes can easily be determined and multiplied with the model matrix. The view and projection

matrix is generated with the described arc-ball algorithm.

If the species of the shown seed is known, because of the feature calculation step or from a loaded feature file, there is also a border around the DVR render widget which matches with the color of the species class in the other plots. If the seed is unknown, the widget gets a black border. The calculation of the color for the different classes is done in the HSL (hue, saturation, lightness) color model. The positive aspect of this model is, that the hue is represented by an angle [IHKM12]. The angle can be easily split into an arbitrary number of colors (Algorithm 5.2).

A negative side of this method is, that the human perception is not taken into account. If many colors are desired, it is hard to distinguish between some of them. The color encoding of the fresh seeds are only lighter versions of the carbonized seed color to symbolize they refer to the same class (see Figure 5.5).

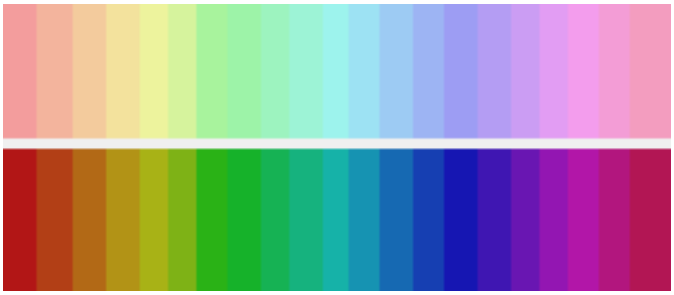


Figure 5.5: A scheme of the used color ranges for the carbonized seeds (upper bar) and the fresh seeds (lower bar). The fresh seeds of the same species get a lighter version of the carbonized ones.

For an overview of all species loaded from the feature file (described in Section 5.2), there are multiple DVR widgets to show a representative seed of each class (fresh and carbonized), if available. These representatives are selected according to the median of the selected features as shown in the next step. The median objects are preferred over mean objects, because they can be found in the data and are not synthetically calculated. Another advantage is that outliers have less influence on the selection process of the representative seeds. It is also possible to select one or more of these widgets to emphasize the appropriate classes in the feature plot. Due to the number of classes, this is an important possibility to get a better view on classes of interests. For a visual registration of the volumes, different 3D methods have been tried, but according to the resolutions of the seed volumes all of them were too slow for real time usage. It was also tried to do the registration on resampled smaller versions, but then the calculations where incorrectly most of the time. For this reason, the registration is done by rotating the volumes with the help of the principal axes. They are rotated to the coordinate axes of the DVR to get a similar orientation to all seeds.

5.2 Feature Extraction

The second sub pipeline containing the feature extraction. A scheme of the pipeline is shown in Figure 5.6.

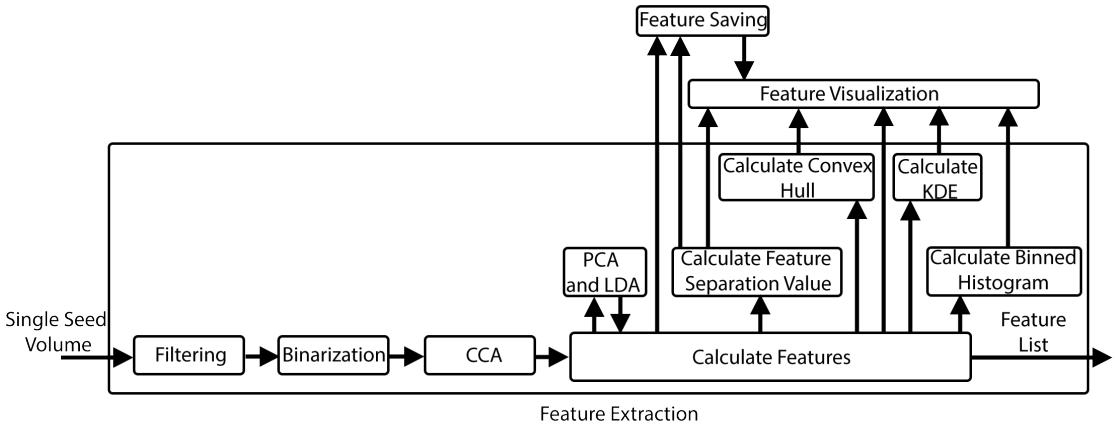
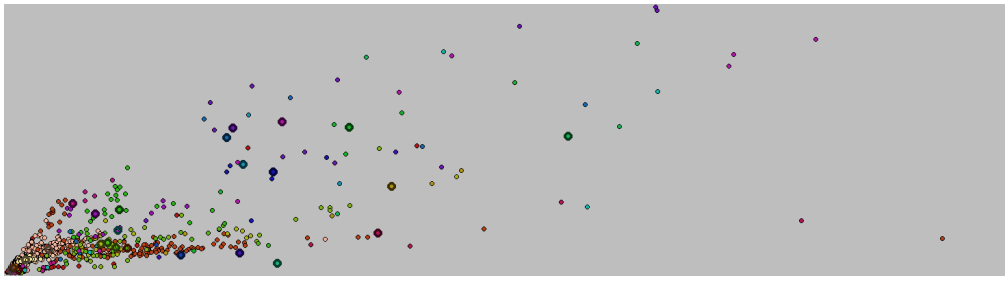
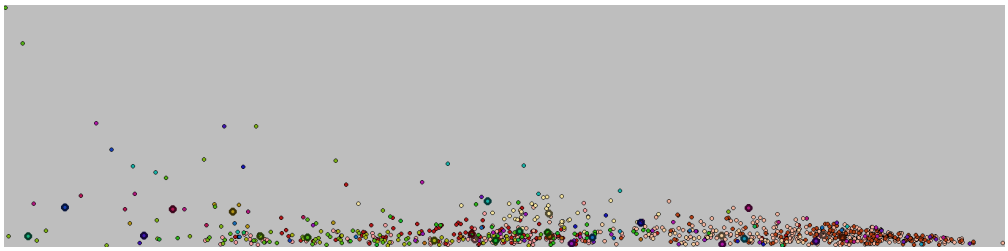


Figure 5.6: The feature extraction pipeline. Since shape features are used, the connected component of the seed has to be calculated. In the following, the features are calculated (partially with the help of the PCA output) and different distribution algorithms, like the Kernel Density Estimation (KDE), are applied for better visualization.

Initially there is also a filtering (median filter explained in Section 3.3.2), a binarization (Section 3.3.2), and a connected component (Section 3.3.2) step. This is necessary because shape features should be calculated and a binary component object is used for the measurements. Since the features should be as accurate as possible, a resolution reduction step for runtime improvements is not possible here. Due to the preprocessing, the Connected Component Analysis (CCA) will return a single component, which is the suggested seed. If there are more than one, the biggest component is selected. The used features are Roundness, Elongation, Equivalent Ellipsoid Diameters (in each direction), Equivalent Spherical Diameter (which is $2 \cdot R_{eq}$), Equivalent Spherical Perimeter (Perimeter is the Surface Area in case of 3D), Surface Area, Flatness, Number of Pixels (refers to the discrete Volume), Physical Size ($volume \cdot voxelsize$), length of the principal axes, Principal Moments (in each direction), ratios (each of the Equivalent Ellipsoid Diameters with each other and each of the length of the principal axes to each other) and the first two PCA Eigenvalues. Texture features were not considered because the surface has a high variance depending on the heat the seeds have been carbonized with, even within the same species. If texture features would be an improvement could be tested in future works. For the Eigenvalues, a PCA (Section 3.3.3) is calculated to get axes with high variances. All points are transformed into the PCA space to visualize the principal components. A Linear Discriminant Analysis (LDA) is also calculated. This method determines an axis which offers a good separation between the classes and is very suitable for classification processes (more details in Section 3.7). By selecting species,



(a) Feature separation value of 0.68. X axis: Surface Area, Y axis: Principal Moment in x direction.



(b) Feature separation value of 0.41. X axis: ratio of Equivalent Ellipsoid Diameter in x and z direction, Y axis: Elongation.

Figure 5.7: Two images of the program with two different feature separation values. One means 100% of the samples are closer to the assigned class median than to the median from the others classes. The median points are marked with a colored ring and 22 species carbonized and fresh are visualized.

the LDA is recalculated for the selected classes. This is be done, because the separation works better with fewer classes. This way, it is possible to successively refine the scatter plot until the unknown seed can be classified. An image of the scatter plot view with LDA enabled can be seen in Chapter 7.

The calculated features are visualized in a 2D scatter plot with selectable features for each axis. To offer the domain expert a good preselection for the initial axes, a feature separation matrix is calculated which contains for each feature pair a separation value. The distances for each point to each median point of the classes are determined. Afterward, the ratio of points which are nearer to the median point of the class they assigned to, and number of all samples is calculated. Since the metric is commutative, only half of the matrix has to be calculated for an optimization. The two features which have the highest number in the matrix are selected for the initial feature plot. An example images for two different separation values are given in Figure 5.7. The detailed procedure is shown in Algorithm 5.3.

Since for each list of feature values a sorted indices list is available, finding the median for the feature separation matrix is only a lookup operation. The minimal and

Algorithm 5.3: Calculate Feature Separation Value

Input: A list which contains lists for each class. These lists contain lists for each feature which contain the feature values ($featureList$)

Output: A two dimensional array which contains the feature separation values between every class pair.

```
1  create a featurecount × featurecount 2D array featureMetricMatrix;
2  for i ← 0 to featurecount − 1 do
3      for j ← 0 to featurecount − 1 do
4          switch do
5              case i = j do
6                  featureMetricMatrix[i, j] = 0;
7                  break;
8              end
9              case i > j do
10                 featureMetricMatrix[i, j] = featureMetricMatrix[j, i];
11                 break;
12             end
13             case i < j do
14                 calculate for each tupel of the featureList[i] and featureList[j] the
                    distance to all class medians (the middle element of the feature
                    index list) of these features;
15                 featureMetricMatrix[i, j] = number of elements with distance to
                    own class median is lower than to the other class median;
16                 featureMetricMatrix[i, j] = featureMetricMatrix[i,j] / number of all elements;
17                 break;
18             end
19         end
20     end
21 end
22 return featureMetricMatrix;
```

maximal values for the axes can be extracted similarly. To give the user an idea how seeds of the species look like, the median seeds are visualized. The median points are also used to load the median seed volumes (if available) for these class representations. The saving process of the volumes to VD files is options, so it might be, that the files of the sees are mot available. Additionally, the median points are marked with colored rings within the plot to show the domain expert the main positions of every class in respective to the selected axes. For a better overview of the class boundaries, a convex hull is calculated and optionally drawn in the scatter plot. For the calculation, the Graham algorithm, explained in Section 3.5, is used. The convex hull also visualize how the class is spread over the plot and outliers can be easily detected. Also as an

optional view, a KDE (Section 3.6) plot can be displayed in the background to have a good visualization of the distribution of the classes. If only one feature is selected, the points of the scatter plot are projected on the feature axis and a binned histogram can be shown instead of the convex hull because a 1D convex hull contains no relevant information. In this view, the distribution of a single feature is visible. This visualization is also useful to detect outliers and to have a local information about the number of elements within a specific range. There is also the option for a 1D KDE plot for a single feature distribution in the background. A binned histogram is also used on each button of the feature selection widget to have an overview of the sample distribution of this feature before selecting it. Screenshots of each of these visualizations are shown in Chapter 6.

A basic interaction in the scatter plot is the selection of data points, which are the seeds. The selection is important to inspect the seeds and to check the different features a seed has. Also the behavior of outliers can be examined. A rectangle tool for multiple selections is also offered. The selected points are marked and labels are assigned with the sample names. To find the samples that should be selected, a binary search is also implemented and used with the sorted lists of the feature values. The selected points can be deleted, to remove outliers, falsely segmented or other problematic seeds, like incorrectly labeled ground truth, which could disturb the classification process. In this case, the PCA and the LDA are also recalculated. If a single point is selected, the seed volume is loaded (if available) and shown in the DVR widget for the selections. In the case of the rectangle tool, a list is offered which contains the percentages of selected elements of each class. In case of the single point selection, the distances of the clicked position to the points have to be checked to determine the closest one. If the axes change, the selected points stay the same. This way, the position of the selected points within the plot can be tracked through the different features. If classes are selected in the volume rendering view for the median seeds of the classes (explained in Section 5.1), all points of the classes are highlighted in the scatter plot and the points of the other classes become more transparent. These class selections are also visible in the convex hulls and KDE view shown in the background of the scatter plot. The colors for the classes are the same as in the borders of the DVR widget for the median seeds. It is also possible to visualize the median objects of each class. The median object is selected because the domain expert can see an existing seed of the dataset and not a synthetic, calculated one, like an average object would be and outliers have less an influence on the selections. This representatives change if the axis selection changes. The DVR visualizations are only possible if the files of the seeds are in the right directory, otherwise they would not be found. If many classes are available, the loading time can be noticeable longer.

The features can be saved into a file (type FLF: Feature List File). Since the calculation of the feature metric matrix took also time, it will also be stored (type FMV: Feature Metric Values). For the use in other programs, the feature list can also be exported into the tabular CSV filetype.

5.3 Machine Learning

In this sub pipeline, a model is trained to get predictions about unknown seeds for the user. The pipeline can be seen in Figure 5.8. Initially, the feature values carbonized and fresh seeds of the same species are merged together. Due to the domain expert, it is not necessary to distinguish between them. It also would not be possible in some classes because of missing fresh samples and, in this way, there are more samples per class after the merging process to train the model. There are some classes which contain only fresh or carbonized seeds and not both.

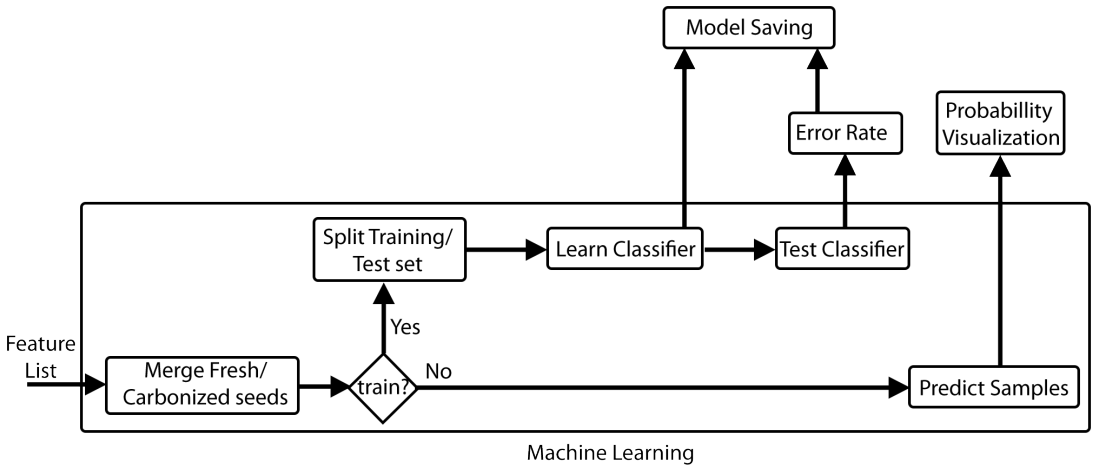


Figure 5.8: The machine learning pipeline used in our system. In this step, new seeds are predicted or a model is trained. The error rate is visualized and the incorrectly classified points are marked to show which classes are more defective. The predictions are listed with probabilities for the classes.

If there is already a trained model and a new sample has to be predicted, the gained probabilities for each class from the classifier are shown into a list to itemize the user which classes are a possibility for the new sample. In the scatter plot, unknown seeds for the predictions are put in a class named 'unknown' and are assigned black as color to be more visible between the other class colors. In case of learning (or relearning) the model, the samples are divided into a training (80%) and a test (20%) set. A voting classifier using a decision tree and a random forest method (Section 3.7) is used because it shows the best results, also with sparse classes. A test of different learners is shown in Chapter 7. The classifier is tested by applying the test dataset treated as unknown. This results in an error rate which describes the ratio of incorrectly classified samples. Typically, the

error is visualized with a confusion matrix (as shown in Chapter 7). In the confusion matrix representation, it is hard to see where the incorrectly classified seeds are located in the scatter plot. In the software, the error is shown by the accuracy value with a green to red color gradient background for visual feedback. Additionally, the incorrectly classified points are marked in the scatter plot to show the user where the critical areas are. If a class is selected and many seeds are marked as misclassified, it can also be determined which classes are especially hard to predict. It is also possible to delete sample points (e.g., outliers) in the scatter plot and retrain the classifier. The changes are directly visible in the error rate and the new incorrectly classified points and it can be seen if the deletion is an improvement of the model. The model can then be saved for future predictions. For a visualization of the error rate without a new cross validation of a loaded machine learner model, the error value is stored within the model as well. In case of the prediction of new samples, a list with all classes is offered which contains the probabilities for each class. The genus names are also displayed to give the domain expert also a suggestion on this systematical level if the species probabilities are very similar.

The aim of these thesis is to combine a machine learning pipeline with visual analytics techniques to to improve the classification process of the domain expert and additionally create a description of the seeds, that is well understandable by biologists and uses the volumetric data available. Since the research field of classification of carbonized seeds is currently very small, there are not many methods adapted for this kind of data. In this work, segmentation and separation methods are adapted and used so can handle the different artifacts and the high number of data concerning to the file size and resolution. Since the feature calculation hardly depends on the quality of the separation, this segmentation methods are very important for the results. On the separated seeds, shape features are selected, that have the best reasonable chance to describe the species and can be used to create scatter plots where the species are distinguishable. Additionally, different classifiers were tested to find one that gets the best accuracy to support the domain expert. To check the results of the classifier and control the labels of the ground truth, different techniques are adapted to the seed data set. In our SEEd software, the seeds can be visually inspected with Direct Volume Rendering (DVR) and based on different shape features in the scatter plot. The scatter plot is selected, because it is well understandable and it can be good enhanced with other techniques to emphasize information within the plot, needed for the tasks of the domain expert.

CHAPTER 6

Implementation

The program is written in C++. For the basic functionality Qt (Version 5.8.0) [Comb] is used and extended for the requirements of the software. The implementation follows the pipeline presented in Chapter 5. The basis of some widgets is taken from the Qt Widgets for Technical Applications (QWT) library [QWT]. QT Charts [Coma] was also tested but fulfills not the necessary requirements because most of the functions can not be overwritten. It could not be adapted to solve the tasks of the software.

For the loading purpose, two different parsing libraries are used. The first one is the DCMTK [OFF] library and provides all paths of the slices belonging to the seed volume and the raw density values from the DICOM files. The TIFF files are parsed with the LibTIFF library [Lib]. Since the VD (volume data), FLF (feature list) and FMV (feature metric) files are self made datatypes, the parsing is done without an additional library. The VD file contains the resolution of the volume, followed by the density values. The FMV file starts with the number of features followed by the elements of the feature separation matrix. In case of the FLF file, it stores first the number of classes (i.e., botanical species). Next, it iterates over all classes and stores the name of the class, the number of elements within the class followed by all feature samples (all feature values, the name, and the subsample number). If there were no subvolumes in the file, the subsample value is -1, otherwise it is the number of the component of the CCA. Saving these files is simmilar to the loading process, but storing the information into the file with the same ordering. It is also possible to export the list of features in a CSV file. As features, there are also the principal axes for the registration. The calculation step of the PCA of the features and the LDA are done with the OpenCV library [Int].

For some parts of the subvolume checking, like the filtering, binarization, resampling, Connected Component Analysis (CCA), and the volume Principle Component Analysis (PCA) the Insight Toolkit (ITK) is used [Kit]. While testing the ITK Otsu Thresholding, the im-

age has to be inverted to get the right results. An alternative implementation of the Otsu Thresholding (Algorithm 6.1) for other parts, like the background filtering in the DVR, is also used because of the additional inverting step needed by using the ITK thresholding.

Algorithm 6.1: Otsu Threshold

Input: A list of integer values ($fValues$), an integer for the number of bins for the histogram ($bins$) and a Boolean to tell the algorithm if the zero values have to be considered.

Output: A float value *thres* which holds the best threshold value towards Otsu.

```
1  hist ← create a histogram of fValues;
2  if zero values are ignored then
3  |  startindex = 1 ;
4  |  size = fValues.size - hist[0];
5  else
6  |  startindex = 0;
7  |  size = fValues.size;
8  end
9  sum, q1, sumB, varMax, thres ← 0;
10 for i ← startindex to bins − 1 do
11 |  sum+ = i ∗ hist[i];
12 end
13 for j ← startindex to bins − 1 do
14 |  q1+ = hist[j];
15 |  if q1 == 0 then
16 |  |  proceed with the next iteration ;
17 |  end
18 |  q2 = size − q1 ;
19 |  sumB+ = j · hist[j] ;
20 |  μ1 = sumB / q1;
21 |  μ2 = (sum − sumB) / q2;
22 |  σ^2 = q1 ∗ q2 ∗ (μ1 − μ2)^2;
23 |  if sigmaSquare > varMax then
24 |  |  thres = j;
25 |  |  varMax = σ^2;
26 |  end
27 end
28 return thres / bins;
```

In case of the DVR part, the OpenGL extensions are used with the help of the GLEW library [GLE]. Some matrix operations are done with the GLM library for better com-

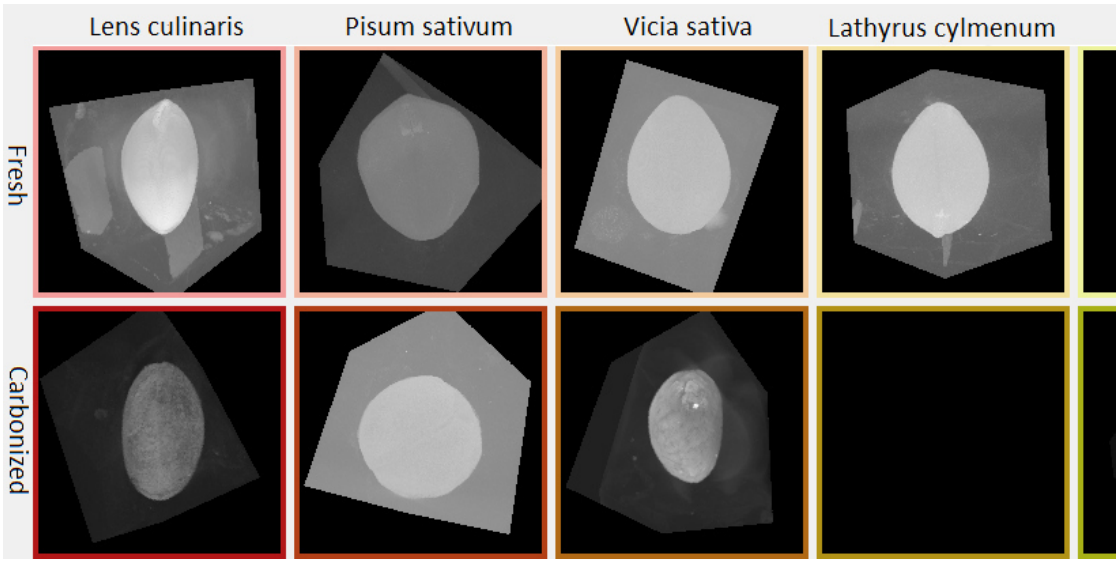


Figure 6.1: An example image showing the widget containing the median seeds of each class. Displayed in this image are only the first four species but all species are represented in this view. In some species only fresh or carbonized samples are available. Render widgets of classes with no samples stay black.

patibility with GLSL shaders [GLM]. As rendering technology, OpenGL is used with the respective shader language GLSL. An image of the different DVR methods used can be seen in Figure 5.4 and the widget to show the selected median seeds of each class is shown in Figure 6.1. The loading time increases with the number of classes. This is why the loading of the median seeds can be deactivated. In this case only the color frames of the seed classes are visible. In case of the alpha compositing method, a transfer function widget is provided. This widget also contains a histogram of the densities from a selected volume (Figure 6.2).

The output of the data loading process is used for the following feature calculation. The result is a list of volume objects, which contains the path to the volume. Additionaly a subseed number to identify the sample if multiple seeds are in the file, the number of elements in each dimension (integer), and list of floats with the densities. Most of the features are calculated with the help of the ITK library. The remaining features are derived from the PCA output of ITK. The output is a list for each class which contains lists for each feature containing all values. Feature values belonging to the same sample are stored on the same index. To speed up some operators (min, max, median, search, ...), there is also a list for each feature which stores the indices of the elements in increasing order as lookup table. Additional information in the class lists are the name of the class and the color of the class. The list of classes are also used to show the number of elements selected by the selection of the rectangle tool (Figure 6.3).

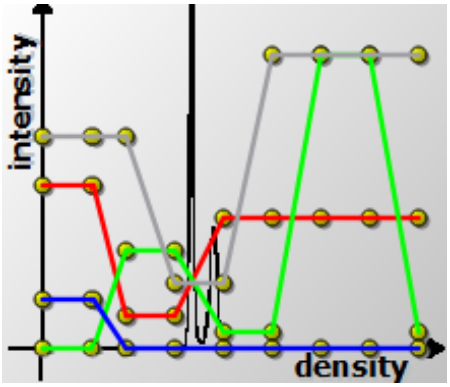


Figure 6.2: Transfer function widget to allow the user to influence the alpha compositing visualization. In the background, a histogram of the densities of a selected volume is shown to support the finding process of the best fitting transfer function.

To offer the user an initial plot, the two best axes should be selected. As metric a quotient of the number of samples closer to the own class median and the number of all samples is used (full explanation in Section 5.1 and in Algorithm 5.3). Since this calculation needs a lot of time, the resulting matrix can be stored in a file (FMV filetype) in the same folder as the feature file. It contains the number of features followed by all values.

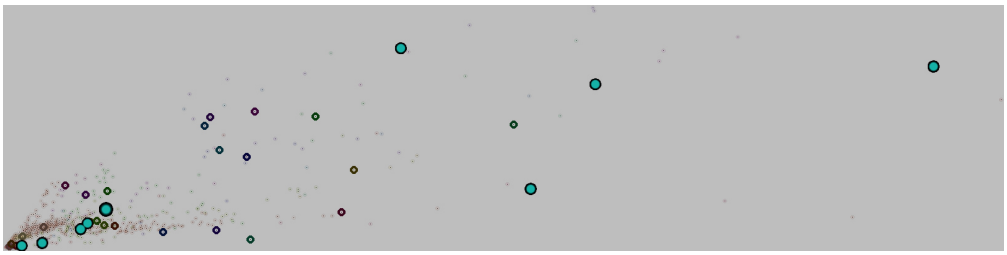
Figure 6.3: A snipped of the software with a selection of the rectangle tool (seeds marked with yellow circles). The black rectangle shows only the selection region. All sample names are visible, and additionally there is a list in the right upper corner that displays the percentage of elements selected per class.

For the KDE (2D for two axes, 1D for a single feature distribution), the Python library Seaborn is used [Was]. KDE is used to show the distributions of the samples in the classes. A C++-to-Python wrapper was written for the communication between these two languages. The data is sent to the Python file with the help of the wrapper, the KDE is calculated there, and sent back to the wrapper, which converts the results into C++ data structures. The result is used as a background for the scatter plot if the user enable the KDE view.

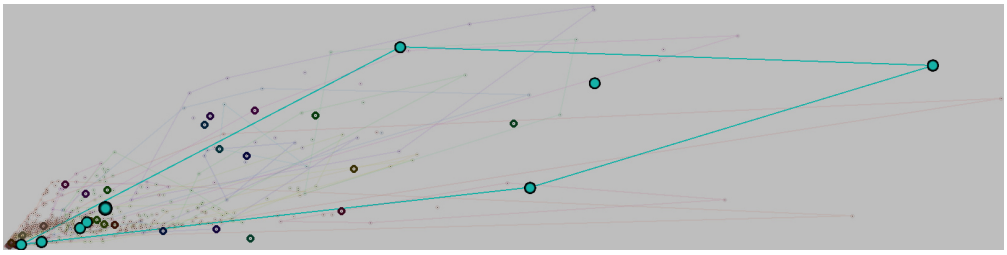
A binned histogram, to show locally the number of samples in specific areas, is calculated. 'Binned' means in this context, that the histogram has a lesser elements than the value range of the data. A bin summarizes a range of values and not only a single one. To visualize such a histogram, a polygon is created which looks like a staircase. This way, the vertical lines of the normally used box plot are not fully connected with the axis, but with the height of the next bin. With many bins, this visualization is clearer for the user. It is used in addition to the KDE plot, to visualize the distribution of the shown data. During the loading process of the feature file, a binned histogram is calculated for each feature to visualize the histograms on the buttons for the feature selection.

It is also possible to show the convex hull around each class in the appropriate color to find outliers fast (spikes in the convex hull) and have an overview of the area the class covers in the plot. This way it is easy to see where and how much the classes overlap. An example image with the different visualizations (KDE, convex hull and binned histogram) within the scatter plot is shown in Figure 6.4.

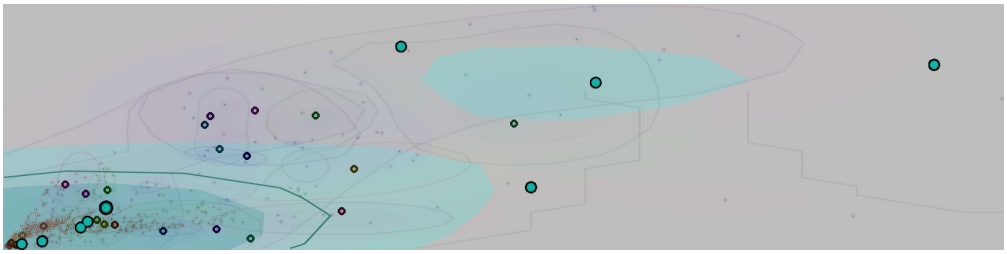
The last step of the pipeline is the machine learning part. Since it offers a wide range of techniques, the Scikit-Learn python library is selected for this classification task [Sl]. Since the software is written in C++, there is also a data conversion wrapper in this step. The model and its accuracy value can be stored into a file and loaded for future predictions. The accuracy is visualized with a number within the toolbox widget whose color depends on how good the value is. In the toolbox it is also possible to change the selection mode from single seed to the rectangle tool. The prediction step sends also the feature list of the unknown seeds and predicting them. The returned result is a list of probabilities for each sample for each class. These probabilities are visualized in the scatter plot by selecting the unknown seeds. The incorrectly classified samples are also displayed in the plot to show the user where most of the uncertainty occurs in the plot and which classes are involved. For this task, the C++ library Dlib [Dli] was also tested, but the training and prediction steps are much slower than the wrapper plus Python approach.



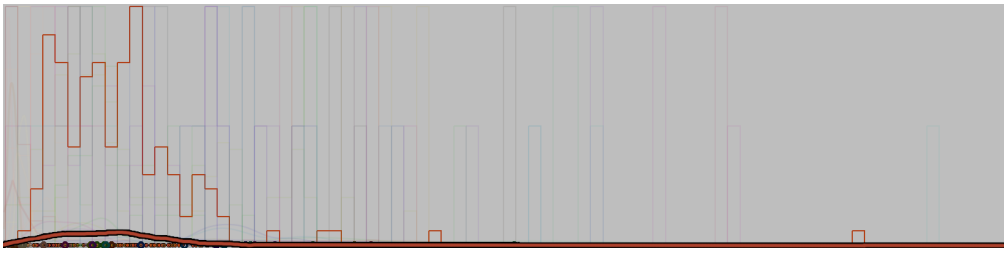
(a) Turquoise class selected. The other cyrcles are the median objects of the other classes.



(b) The convex hull of the selected turquoise class. Outliers are visible (e.g., the sample near the right border).



(c) Turquoise class selected with KDE plot. The distribution of the samples of the class is visible as shaded colors (three levels) and the areas with the highest densities are marked with distribution lines (turquoise circle).



(d) 1D distribution view. All points are projected on the x axis. The KDE (orange curve) to show the overall distribution and a binned histogram to emphasize outliers (e.g., the most right sample) and to visualize local maxima (left in the plot). The KDE curve is normalized over the number of samples to keep the proportions and the binned histogram is normalized between zero and one.

Figure 6.4: Different additional visualizations on the scatter plot are shown. X axis: Surface Area, Y axis: Principal Moment in x direction. In the (d) the Y axis is between zero and one for the binned histogram and between zero and the number of all samples in case of the KDE plot. The selected samples are highlighted for the figure to have a better contrast to the background.

CHAPTER 7

Results

The main goal of this thesis was the creation of a software to classify carbonized seeds. Since an automated classifier accuracy was expected as non optimal, additional visualizations are offered to cross check the results of the machine learner. The outcome of the classifier can be seen at the end of this chapter because the results are presented in pipeline order. The runtimes are tested on a PC with the following properties: Windows 10, 64 bit, Intel i5-7500 CPU, 16 GB DDR4 RAM and a NVIDIA GTX 1070 (8 GB GDDR5).

Before the other results of the software, the low level user tasks described in the beginning of Chapter 5 are analyzed. The results for the first task, i.e., loading all necessary data types, is described in the next paragraph. The next user task is the inspection of the data. The different Direct Volume Rendering (DVR) methods to inspect the seed volume directly can be seen in Figure 5.4. The overview widget of the Computed Tomography (CT) data for all botanical species which displays their median seeds, is shown in Figure 6.1. Figure 6.4 contains different additional visualizations to find clusters of seeds, main regions of samples of a botanical species respective to a feature and outliers. This outliers can be incorrectly labeled ground truth, incorrectly segmented seeds, uncommon members of the species or scanning errors (described later). The outliers can be deleted which may improve the classification of unknown seeds. The additional visualizations can also be used to determine in which region of specific features an unknown seed is located and to which species this seed could be refer. The next user task is the creation of a descriptor. The used shape features can be exported to an Comma Separated Values (CSV) file. The last user task is the classification. Sample classifications and an analysis of the machine learners are displayed later in this chapter.

Initially, the results of the loading step are described. The first results noted are the volumes with the highest memory usage per datatype within the actual data samples,

followed by the average filesize and runtime. It has to be noted, that, since every voxel has to be loaded, the loading time increases linearly with the filesize and takes approximately two minutes for a resolution of $1300 \times 1125 \times 1100$ (which are over 1.6 billion values) in DICOM format (~3 GB space). The average memory size of the DICOM files is 2 GB with a loading time of 1.5 minutes. With an added seed separation step, the runtime stays under three minutes. In VD format, the loading of a ~850 MB file needs approximately three seconds ($594 \times 570 \times 642 \approx 217$ million values). The average space and time consumption lies by 200 MB and 1 second. Since the VD files are created from TIFF and DICOM files, the variance of memory usage is very high. A 82 MB TIFF file with a resolution of $388 \times 432 \times 202$ (~33 million values) has a runtime of ~2 seconds (average: approximately 30 MB and 1 second). In the given dataset, the seeds in samples containing multiple seeds could nearly all separated by our software. The error rate of the segmentation step is under 1%. In the incorrectly separated samples, the seeds were very close to each other. During the resampling too small gaps between the samples disappear and multiple seeds merge to one connected component. This seeds could be proper segmented if the downsampling rate is decreased. Since the user can delete outliers, it is better to detect a component that is not a seed than rejecting an actual seed. Because of this case, the filters of the connected components has more tolerances. In case of the available dataset, this was not necessary but since the scanning process of the domain expert is not finished yet, it could be necessary in the future. In the DVR, an Otsu Thresholding is used to filter the background. There are a few samples in which the threshold can not be determined correctly because of the background noise pattern, and the visualization would be all black. For this reason, the MIP technique has no background filtering, so this visualization always shows the data properly. The threshold is only used for the visualization and does not affect the feature calculation. The saving process in the VD format reduces the needed space compared to the DICOM format but compared to the TIFF format it expands approximately by five MB.

The feature calculation takes approximately one minute (with a resolution of $546 \times 630 \times 546$) depending on the resolution. The bottleneck in this step is the single selection because the runtime scales linearly with the sample count. The rectangle tool has logarithmic time because of the binary search. All other calculations are only done once. The calculation of the feature separation matrix takes approximately 30 seconds with the available dataset. If the number of classes and samples grows, this is a possible slowdown of the loading process. This is why a saving option of the feature separation matrix is available. The binned histogram, KDE, and the convex hull have only be calculated if an axis changes. A process which needs much time is the loading of the median seeds of every class for the overview DVR plot. In the worst case there are two loading operations (for the fresh seeds and carbonized seeds) for each class every time an axis changes. With three botanical species, where fresh and carbonized seeds are available, the loading time is approximately one second. With 22 species, the loading time increases to nearly one minute. Because of this, the plots for the median volumes can be deactivated by the domain expert for faster axes changes. The automatic calculation of the features

and separation of the seeds, combined with the export into a tabular file format was considered as very useful by the domain expert to add the feature descriptors directly into the seed database. During the examination of the outliers there were also files detected that have CT scanning errors. This is also very useful for the domain expert, because these seeds have to be rescanned (see Figure 7.1).

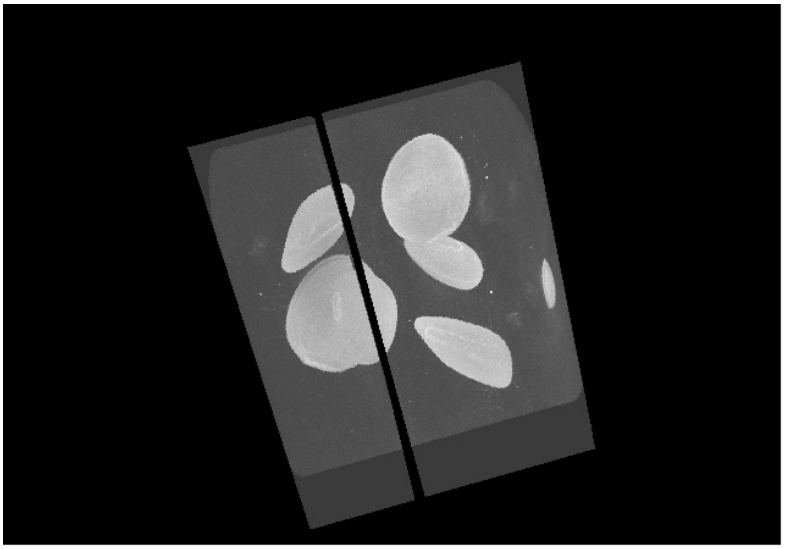


Figure 7.1: An image with CT scanning errors found by the examination of the outliers.

For the machine learning part, many learners methods were tested with various parameters and the accuracy are compared. Table 7.1 displays the best outcome per learner with the used parameters which are different from the standard parameters of Seaborn [Was]. These results in the selection of the voting-classifier containing Random Forest and Decision Tree learners with soft voting. Since the accuracy of neuronal network differs is not very high from the voting approach, maybe this classifier could be better if more samples are available in the future. The seed scanning project is still ongoing and a much greater dataset will be used later. Figure 7.2 contains the confusion matrices of the used voting classifier and the neuronal network which contain the number of correctly classified samples in the diagonal. The incorrectly classified ones are in the row of the class they confuses.

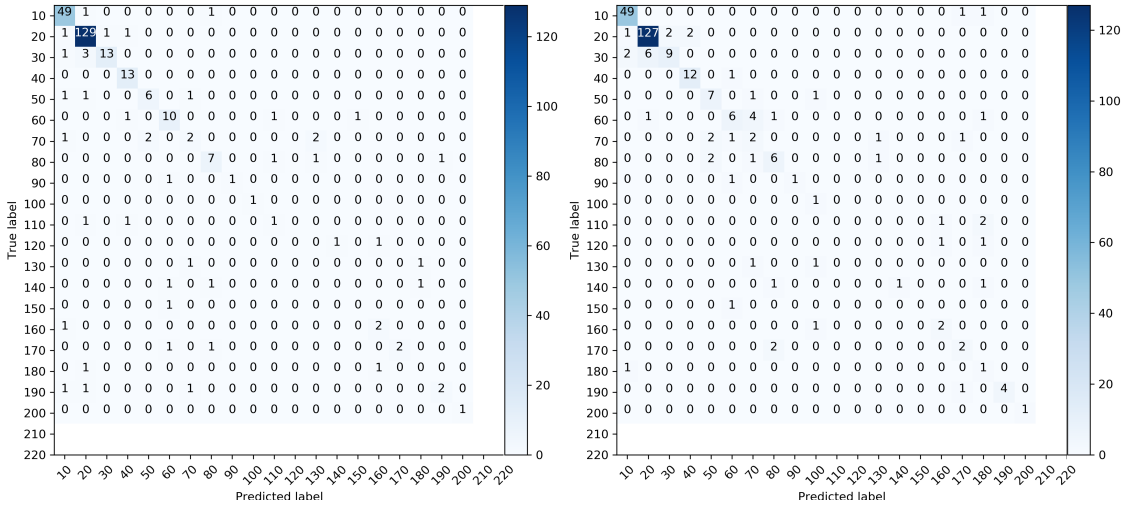
The classifiers are more accurate than initially assumed but that increases the usability of the software for the user. Because of the different appearances of seeds of the same species due to the carbonation on different heats and other physical deformations, additional to the unbalanced classes, the accuracy of the classifier was assumed as lower. It could be considered to use the learner as full automatic solution but as seen in the diagonal of the confusion matrix, there are classes that have a low accuracy due to the low number of samples. Especially in seeds of these classes, the domain expert can use

Machine Learner	Parameter	Accuracy
Gaussian Naive Bayes	-	52,23%
AdaBoost	estimators = 100, learning rate = 0.2	63,35%
Gradient Boost	estimators = 200, learning rate = 0.2, max depth = 1	79,36%
Decision Tree	min samples split = 3	84,67%
Random Forest	estimators = 40	83,27%
Bagging (K-Nearest Neighbor)	max samples = 0.5, max features = 0.5	68,33%
Voting classifier (Random Forest + Gaussian Naive Bayes)	voting = hard	76,51%
Voting classifier (Random Forest + Decision Tree)	voting = soft	85,05%
Neuronal Network (Multilayer Perceptron) with data normalization	hidden layer sizes = (30, 30, 30)	82,21%
Linear Discriminant Analysis	-	76,51%

Table 7.1: Accuracies and parameters of different machine learner methods tested on the calculated features. Since many classifiers use randomness in some way, the accuracies may vary in a small range.

the given seed measurements and visualizations to determine their correct classes. If the incorrectly classified samples are examined in the scatter plot, there are many samples which the user can assign to the correct class. Figure 7.3 shows an example for this case.

For a good axis selection, the most important features of the learner could also be examined. Since the used learner is a voting classifier containing other learner methods, it is not possible to show directly the feature importances. If the random forest and the decision tree are evaluated separately (the voting classifier only combines the results of these two), the most important features can be visualized. In case of the decision tree, the ratio between the x and z direction of the Equivalent Ellipsoid Diameter and the Equivalent Ellipsoid Diameter in x direction are the two main features. If these two features are visualized in the scatter plot it can be seen, that the classes have a good



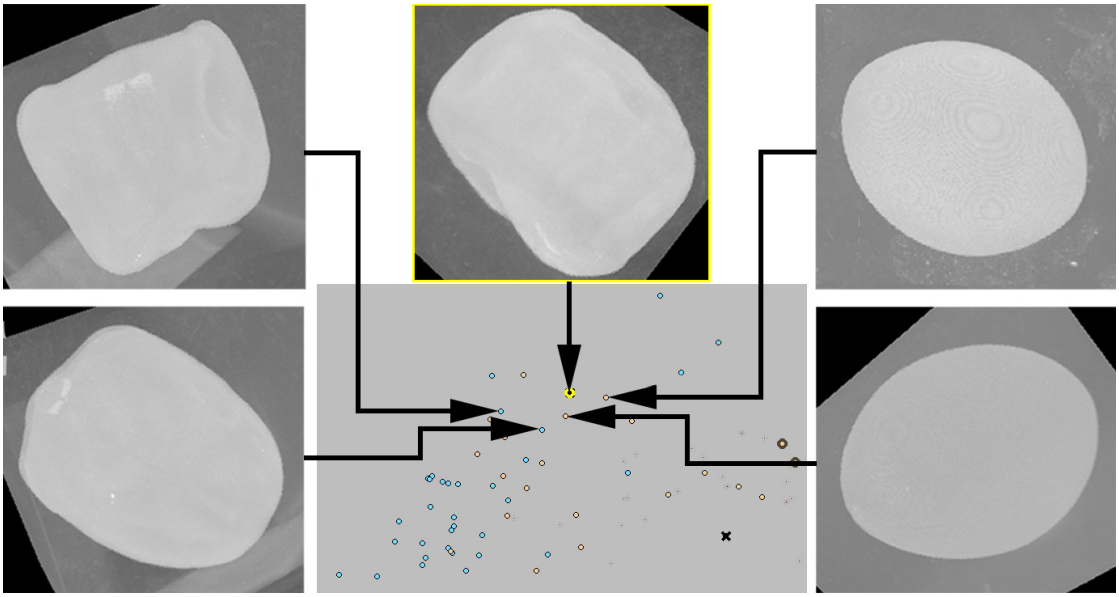
(a) Voting (Descision Tree + Random Forest). (b) Neuronal Network.

Figure 7.2: The confusion matrices of the used voting classifier and the neuronal network which may be interesting in the future because of the increasing number of samples. Values in the diagonal are the correctly classified sampled.

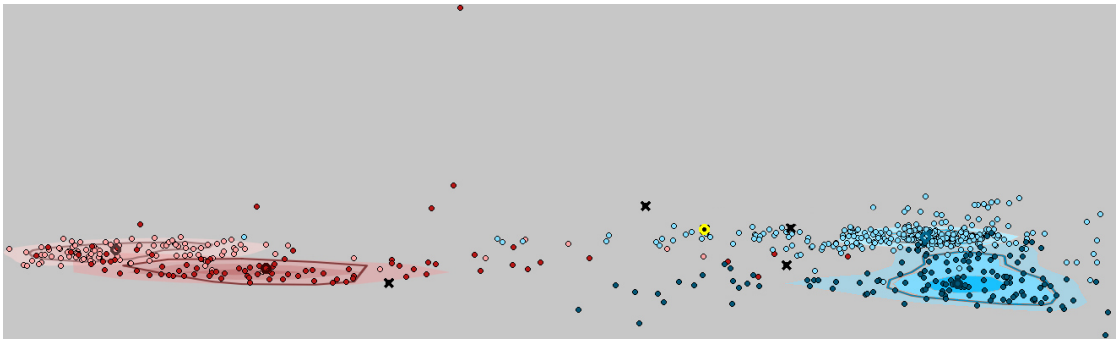
degree of separation. In case of the random forest, the two most important features are the Flatness and also the ratio of the Equivalent Ellipsoid Diameter in x and z direction, but the differences in importance to the other features are very low (approximately 0.01). In the visualization, these two axes are also not visibly separated. The reason for this result is that the random forest contains 40 separate classifiers which have different importances each. Figure 7.4 shows the five most important features for these two classifiers and the scatter plot of the top two features.

In case of the prediction of unknown seeds, there is also a list visualized on the scatter plot that holds the probabilities of each species. The KDE plot and/or the convex hull in the background of the scatter plot can be deactivated if it is no help on a specific sample. If the data features are inspected in the scatter plot it can be seen that an exclusively manual classification without the machine learning part can be very time consumption in some cases because the classes are not completely separated in every view. For seeds in overlap regions the right feature combination has to be found for the axes. The LDA can be a good help in these cases, but is not a guarantee to manually classify every unknown seed.

It has to be noted that a preselection of possible species is often sufficient because the expert user has additional information on the species and can directly exclude some of the candidates. Figure 7.5 shows an example classification. In the plotted probability list in the right upper corner, the genus names are also visible to give the domain expert the possibility to determine at least the genus of the unknown seed. During some species of the gathered seeds are already extincted or not scanned yet, the classification on the



(a) Samples which can be classified by a human through visual inspection of the seeds. The seed in the top center is the 'unknown' seed, the left column contains *Pisum sativum* (the correct species) and the right column contains *Vicia sativa* seeds close to the unidentified seed.

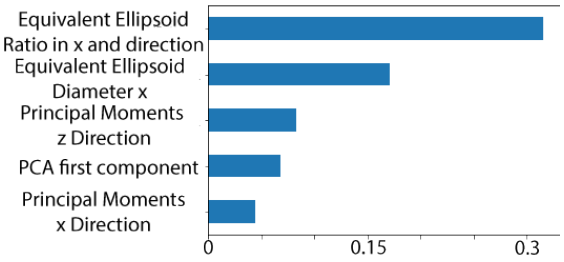


(b) Samples which could be classified with the help of the LDA view (the two classes with the highest probability selected). The axes are the first and second LDA components.

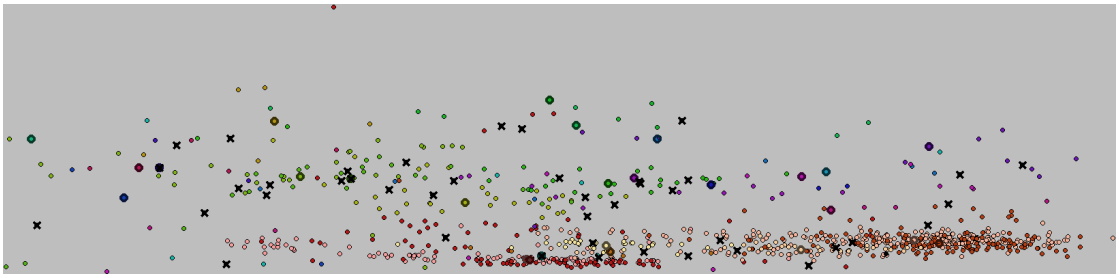
Figure 7.3: An example of incorrectly classified samples of the machine learner (marked with back crosses) which can be correctly classified by a user manually. The color of the *Pisum sativum* class is changed to blue for a better contrast.

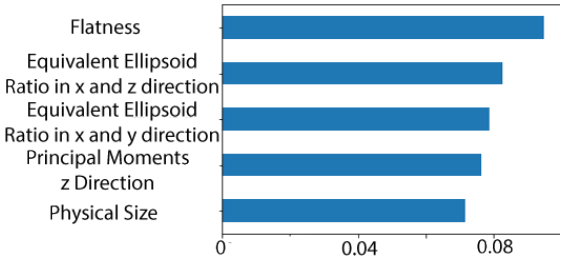
genus level is also important. On genus level, the used classifier has an accuracy of approximately 90%.

There is also geographic information available which can be used to sort out some candidates because often species are located on specific landscapes. This information was not stored on a central place for the use as a feature yet.



(a) Most important features of the decision tree classifier (left). The best features are the Equivalent Ellipsoid Ratio of the x and z direction and Equivalent Ellipsoid Diameter x used as axes in the plot below). Samples of same color are roughly separated.





(b) Most important features of the random forest classifier (left). The best features are the Flatness and the Equivalent Ellipsoid Ratio of the x and z direction used as axes in the plot below). Samples of same color are not so well separated as in the decision tree.

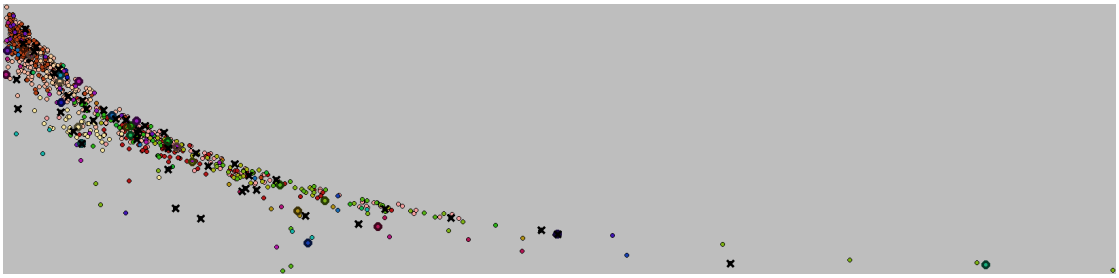


Figure 7.4: Plots of the most important features of the used voting classifiers. The samples in the random forest scatter plot looks less separated because it contains several classifiers with different feature importances.

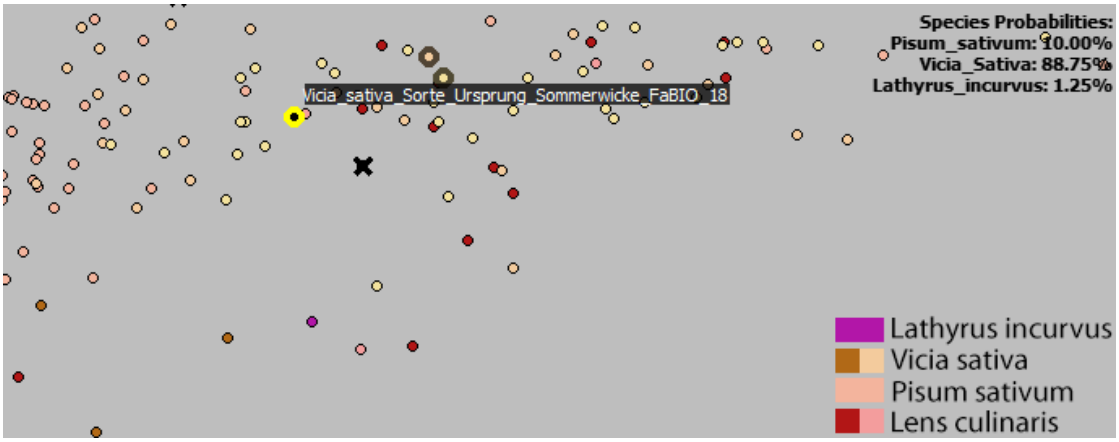


Figure 7.5: An example classification of the marked sample (black with yellow circle) is shown. Incorrectly classified samples of the machine learner are marked with black crosses. In the right upper corner is the output list of the learner with all classes with a probability greater than zero. The genus names are also included, to offer a possibility to determine at least the genus of the unknown seed. *Vicia sativa* is the correct species of the unknown sample. X axis: Surface Area, Y axis: Principal Moment in x direction

Before this work, it was hard for the domain expert to check the classifications of the external expert that manually labeled them. They also tried to extract some significant features of the seeds by hand. Although there is a high variance of appearances of the seeds of the same species because of the carbonization process and other physical performances, the selected features visualized in the scatter plot allow differentiate the species in a sufficient degree. The only technique, the domain expert considered as not useful, is the alpha compositing. This is because of finding the appropriate transfer function for the seeds is not an easy task. This could be improved by adding a method that finds automatic the best fitting transfer function. The SEEd software was commented as very useful by the domain expert. He works with the SEEd software and elaborates on a concept to integrate it further with the processes of the international project of the seed database. An image of the user Interface of the software is given in Figure 7.6.

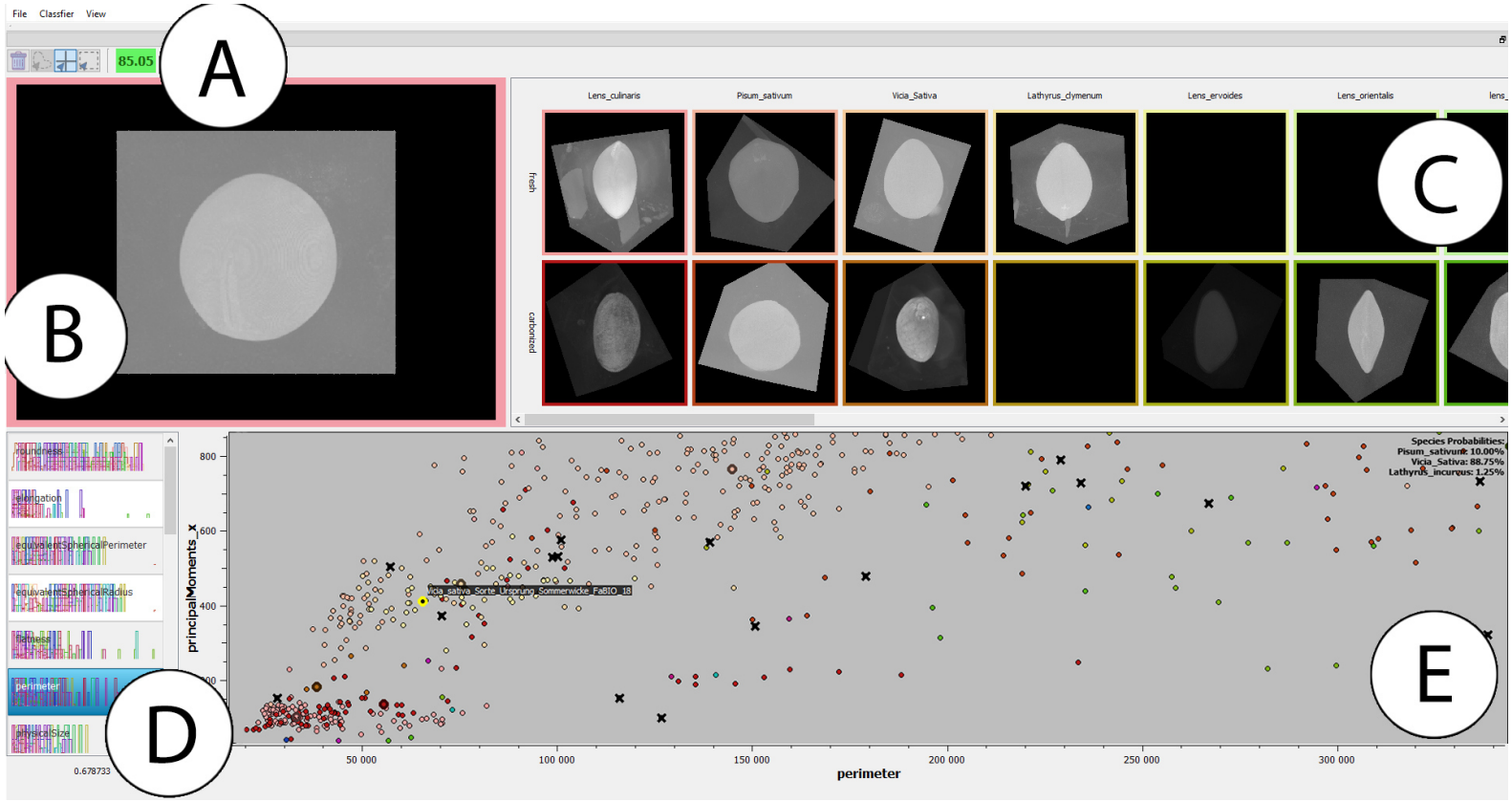


Figure 7.6: A screen shot of the final software. The accuracy of the classifier can be seen in the toolbox widget, next to the selection tools and the deletion button (A). Below, a DVR widget for selections (B) and the widged with the median seeds of all species within the model (C) are shown. In the left lower corner the feature selection list can be seen (D). The scatter plot of the features is next to this list (KDE and convex hull deactivated) (E).

CHAPTER 8

Conclusion and Future Work

Several features of the software, like the automatic seed separation, feature calculation and the ability to export the features to a tabular file format, and the accurate learner, are marked as very useful by the expert user. The separation is done with a mask created due to a binarization step followed by a Connected Component Analysis (CCA). Partial seeds and other artifacts are filtered out by consulting their number of pixels and their distance to the border. For the distribution visualization, Kernel Density Estimation (KDE) and convex hull calculations are used. As additional techniqued a Principle Component Analysis (PCA) and a Linear Discriminant Analysis (LDA) is used. The LDA can be refined by selected specific species. For the classification of an unknown seed, a list is offered which contains the probabilities of all classes. To have at least a classification of the genus, the genus names are also visible in this list.

The results have shown that with the selected shape features, the voting classifier achieves an accuracy of approximately 85%, which is better than expected. The expectations lower than 85% come from the fact that the shapes and textures of the sample seeds are distorted due to the carbonization process or other physical forces. The used voting classifier contains a random forest and a decision tree method, which are combined with a soft voting technique. If the species classification is not sufficient enough, the displayed genus names can be used to get at least a classification on this systematical level. The neural network has also a good accuracy. If there are more samples in the future, as the seed scanning project is still ongoing, maybe this classifier will get even better, because these neuronal networks works best with a big amount of test data. The most important shape features can be visualized and the correlation between different shape features can also be seen to show which parameters are good descriptors of a specific class. The detection of outliers is also a useful possibility. Outliers in the shape features could be incorrectly segmented or initially incorrectly labeled seeds. Selected seeds stay selected if the axes are changed. This way it is possible to validate which of the shape features differ

mostly from other seeds of the same class. Incorrectly classified seeds are directly visible in the plot and problematic classes with low accuracy and ranges within the features which are critical can be determined. All of these points show that the software is a good improvement of the current classification pipeline. It may be that the histograms assigned to the buttons of the feature selection got very confusing as the number of classes increases. This element of the software could be removed in the future.

There are some questions left for future work. Initially, it has to be tested if the volumetric data is an advantage for the learner. Many projects in these field use 2D images to extract the features which would be much faster (the acquisition is also cheaper). It would be possible to use the principal axes to determine the index of slice in each direction where the seed has the biggest size and resample the volume on that index to get the necessary slices. With a threshold and morphological operators (opening and closing), the shape could be determined. Another improvement could be the use of texture features. Since the texture changes with the amount of heat during the carbonization process, they are not considerd in this work. The domain expert had the idea that the size and shape of the pores within the seed could be a feature to differentiate the seed species. The expert also presented a prospect of geographic information about each sample as some species only grow in specific areas. This can also be used to filter out species candidates during the classification process. With an emphasis on the visualization, it has to be tested if it is better to copy the voxels of a seed (with the help of the CCA) on a black background instead of filtering of the background. This could also remove some of the artifacts before the feature extraction. Another question concerning the visualization is the differentiation of carbonized and fresh seeds in the plot. This could help to check the differences between them in the plot. Since the domain expert is also interested on how the carbonization changes the shape of the seeds, this is not done in the current state of the software. For the classification, they are merged together because the fresh seeds should be used with the carbonized ones, to get a broader ground truth. If the fresh and carbonized samples of the same species would be merged in the scatter plot as well as for the classifier, it would also help with a better contrast between the class colors because the number of different colors decreases. Also color blindness is a property which has to be considered more in the future. To determine the genus of a plant, the listed species can be used, because the genus name is assigned to every seed name. It would also be possible to train a classifier on the genus level to get also a classification on this level. And last, there are recent Convolutional Neural Network (CNN) classifiers, which can be trained with very few samples. If they are sufficient for this task, also has to be tested

List of Figures

Acronyms

AABB Axis Aligned Bounding Box. 37, 38

AAM Active Appearancece Model. 35

ANN Artificial Neural Network. 33, 44, 45

ASM Active Shape Models. 35

CART Classification and Regression Trees. 26, 27

CCA Connected Component Analysis. 11–14, 50, 51, 54, 61, 73, 74

CCL Connected Component Labeling. 12

CNN Convolutional Neural Network. 33, 35, 44, 45

CPU Central Processing Unit. 67

CSV Comma Separated Values. 47

CT Computed Tomography. xi, xiii, 4, 7, 8, 17, 45, 49

DICOM Digital Imaging and Communications in Medicine. 8, 49, 50, 55, 61, 67

DVR Direct Volume Rendering. 2, 35, 37, 47, 48, 51–53, 56, 57, 62, 67, 68, 71, 75

GLCM Grey Level Co-occurrence Matrix. 20

GPU Graphics Processing Unit. 40

IQR Inter Quartile Range. 23

ITK Insight Toolkit. 62

KDE Kernel Density Estimation. 2, 22, 24, 25, 56, 57, 64, 66, 68, 69, 71, 73

kNN k-Nearest-Neighbor. 25, 29, 32, 75

LDA Linear Discriminant Analysis. 28, 43, 45, 46, 54, 56, 61

MIP Maximum Intensity Projection. 41, 51, 52, 67

OBB Oriented Bounding Box. 37, 38

PCA Principle Component Analysis. 7, 14–16, 28, 44, 53, 54, 56, 61, 62, 75

QWT Qt Widgets for Technical Applications. 61

ReLu Rectified Linear Unit. 33, 34

TIFF Tagged Image File Format. 8, 49, 61, 67

UI User Interface. 48

wkNN Weighted k-Nearest-Neighbors. 25, 26

Bibliography

[AB11] Ferran Antolín and Ramon Buxó. Proposal for the systematic description and taphonomic study of carbonized cereal grain assemblages: a case study of an early Neolithic funerary context in the cave of Can Sadurní (Begues, Barcelona province, Spain). *Vegetation History and Archaeobotany*, 20(1):53–66, 2011.

[AHH+14] Bilal Alsallakh, Allan Hanbury, Helwig Hauser, Silvia Miksch, and Andreas Rauber. Visual methods for analyzing probabilistic classification data. *IEEE transactions on visualization and computer graphics*, 20(12):1703–1712, 2014.

[AQIS07] Ayman AbuBaker, Rami Qahwaji, Stan Ipson, and Mohmmad Saleh. One scan connected component labeling technique. In *2007 IEEE International Conference on Signal Processing and Communications*, pages 1283–1286. IEEE, 2007.

[Ass92] Adobe Developers Association. Tiff 6.0 Specification. *http://partners. adobe. com/public/developer/tiff/index. html*, 1992. Accessed: 2020-09-02.

[BH00] Imad A Basheer and Maha Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, 43(1):3–31, 2000.

[BJPA09] Terence A Brown, Martin K Jones, Wayne Powell, and Robin G Allaby. The complex origins of domesticated crops in the Fertile Crescent. *Trends in ecology & evolution*, 24(2):103–109, 2009.

[BJT08] Gleb Beliakov, Simon James, and Luigi Troiano. Texture recognition by using GLCM and various aggregation functions. In *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence)*, pages 1472–1476. IEEE, 2008.

[BN16] Juan Pablo Balarini and Sergio Nesmachnow. A C++ implementation of Otsu's image segmentation method. *Image Processing On Line*, 6:155–164, 2016.

[Bre96] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[Cab] Cabi. Lens culinaris sheet. `https://www.cabi.org/isc/datasheet/30274`. Accessed: 2020-03-20.

[CCL04] Fu Chang, Chun-Jen Chen, and Chi-Jen Lu. A linear-time component-labeling algorithm using contour tracing technique. *computer vision and image understanding*, 93(2):206–220, 2004.

[CET99] Timothy F. Cootes, Gareth J. Edwards, and Christopher J Taylor. Comparing active shape models with active appearance models. In *Bmvc*, volume 99, pages 173–182. Citeseer, 1999.

[CET01] Timothy F. Cootes, Gareth J. Edwards, and Christopher J. Taylor. Active appearance models. *IEEE Transactions on pattern analysis and machine intelligence*, 23(6):681–685, 2001.

[CFWJ15] Michael Charles, Emily Forster, Michael Wallace, and Glynis Jones. "nor ever lightning char thy grain" 1: establishing archaeologically relevant charring conditions and their effect on glume wheat grain morphology. *STAR: Science & Technology of Archaeological Research*, 1(1):1–6, 2015.

[CLKP10] Jaegul Choo, Hanseung Lee, Jaeyeon Kihm, and Haesun Park. ivisclassifier: An interactive visual analytics system for classification based on supervised dimension reduction. In *2010 IEEE Symposium on Visual Analytics Science and Technology*, pages 27–34. IEEE, 2010.

[CM11] Angela Cantatore and Pavel Müller. *Introduction to computed tomography*. DTU Mechanical Engineering, 2011.

[CMS16] Emilio Cervantes, José Javier Martín, and Ezzeddine Saadaoui. Updated methods for seed shape analysis. *Scientifica*, 2016, 2016.

[Coma] The QT Company. Qt charts framework. `https://doc.qt.io/qt-5/qtcharts-index.html`. Accessed: 2020-03-20.

[Comb] The QT Company. Qt framework. `https://www.qt.io/`. Accessed: 2020-03-20.

[Coo00] Tim Cootes. An introduction to active shape models. *Image processing and analysis*, 243657:223–248, 2000.

[DFB19] Anne-Sophie Dirand, Frédérique Frouin, and Irène Buvat. A downsampling strategy to assess the predictive value of radiomic features. *Scientific reports*, 9(1):1–13, 2019.

[DGBCNÁVA17] Jesús Díaz García, Pere Brunet Crosa, Isabel Navazo Álvaro, and Pere Pau Vázquez Alcocer. Downsampling methods for medical datasets. In *Proceedings of the International conferences Computer Graphics, Visualization, Computer Vision and Image Processing 2017 and Big Data Analytics, Data Mining and Computational Intelligence 2017: Lisbon, Portugal, July 21-23, 2017*, pages 12–20. IADIS Press, 2017.

[Dli] Dlib. Dlib. `http://dlib.net/`. Accessed: 2020-03-20.

[Ebe06] David Eberly. *3D game engine design: a practical approach to real-time computer graphics*. CRC Press, 2006.

[FSA99] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[FTU95] F. Feito, Juan Carlos Torres, and A. Urena. Orientation, simplicity, and inclusion test for planar polygons. *Computers & Graphics*, 19(4):595–600, 1995.

[GHT17] Xiaohong W Gao, Rui Hui, and Zengmin Tian. Classification of CT brain images based on deep learning networks. *Computer methods and programs in biomedicine*, 138:49–56, 2017.

[GLE] GLEW. OpenGL Extension Wrangler Library. `http://glew.sourceforge.net/`. Accessed: 2020-03-20.

[GLM] GLM. OpenGL Mathematics. `https://glm.g-truc.net/0.9.9/index.html`. Accessed: 2020-03-20.

[GY83] Ronald L Graham and F Frances Yao. Finding the convex hull of a simple polygon. *Journal of Algorithms*, 4(4):324–331, 1983.

[HCS12] Lifeng He, Yuyan Chao, and Kenji Suzuki. A new two-scan algorithm for labeling connected components in binary images. In *Proceedings of the World Congress on Engineering*, volume 2, 2012.

[HK76] Joseph Hoshen and Raoul Kopelman. Percolation and cluster distribution. I. Cluster multiple labeling technique and critical concentration algorithm. *Physical Review B*, 14(8):3438, 1976.

[HMEE14] Badr Hssina, Abdelkarim Merbouha, Hanane Ezzikouri, and Mohammed Erritali. A comparative study of decision tree ID3 and C4. 5. *International Journal of Advanced Computer Science and Applications*, 4(2):13–19, 2014.

[Ho95] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[HOAJ+10] Hadzli Hashim, Fairul Nazmie Osman, Syed Abdul Mutalib Al Junid, Muhammad Adib Haron, and Hajar Mohd Salleh. An intelligent classification model for rubber seed clones based on shape features through imaging techniques. In *2010 International Conference on Intelligent Systems, Modelling and Simulation*, pages 25–31. IEEE, 2010.

[HRZZ09] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

[HS04] Klaus Hechenbichler and Klaus Schliep. Weighted k-nearest-neighbor techniques and ordinal classification. *discussion paper*, 399, 01 2004.

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[IHKM12] Noor A Ibraheem, Mokhtar M Hasan, Rafiqul Z Khan, and Pramod K Mishra. Understanding color models: a review. *ARPN Journal of science and technology*, 2(3):265–275, 2012.

[Int] Intel. OpenCV. `https://opencv.org/`. Accessed: 2020-03-20.

[JC16] Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065), 2016.

[Kan11] Mehmed Kantardzic. *Data mining: concepts, models, methods, and algorithms*. John Wiley & Sons, 2011.

[Kit] Kitware. Insight toolkit. `https://itk.org/`. Accessed: 2020-03-20.

[KW03] Jens Krüger and Rüdiger Westermann. Acceleration techniques for GPU-based volume rendering. In *IEEE Visualization, 2003. VIS 2003.*, pages 287–292. IEEE, 2003.

[KZP07] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

[Lib] LibTIFF. LibTIFF. `http://www.libtiff.org/`. Accessed: 2020-03-20.

[LKG+16] Patric Ljung, Jens Krüger, Eduard Gröller, Markus Hadwiger, Charles D Hansen, and Anders Ynnerman. State of the art in transfer functions for direct volume rendering. In *Computer Graphics Forum*, volume 35, pages 669–691. Wiley Online Library, 2016.

[LL12] Gaëtan Lehmann and David Legland. Efficient N-dimensional surface estimation using Crofton formula and run-length encoding. *Efficient N-Dimensional surface estimation using Crofton formula and run-length encoding, Kitware inc (2012)*, 2012.

[LMS+19] S.K. Lakshmanaprabu, Sachi Nandan Mohanty, K. Shankar, N. Arunkumar, and Gustavo Ramirez. Optimal deep learning model for classification of lung cancer on CT images. *Future Generation Computer Systems*, 92:374–382, 2019.

[LP11] Benjamaporn Lurstwut and Chomtip Pornpanomchai. Plant seed image recognition system (PSIRS). *International Journal of Engineering and Technology*, 3(6):600, 2011.

[LZB08] Rong Liu, Hao Zhang, and James Busby. Convex hull covering of polygonal scenes for accurate collision detection in games. In *Graphics Interface*, pages 203–210, 2008.

[LZO06] Tao Li, Shenghuo Zhu, and Mitsunori Ogihara. Using discriminant analysis for multi-class classification: an experimental investigation. *Knowledge and information systems*, 10(4):453–472, 2006.

[Mar63] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[MDG08] Mario Mustra, Kresimir Delac, and Mislav Grgic. Overview of the DICOM standard. In *2008 50th International Symposium ELMAR*, volume 1, pages 39–44. IEEE, 2008.

[Meb07] Johan Ernest Mebius. Derivation of the Euler-Rodrigues formula for three-dimensional rotations from the general formula for four-dimensional rotations. *arXiv preprint math/0701759*, 2007.

[MMH+17] Yuma Miki, Chisako Muramatsu, Tatsuro Hayashi, Xiangrong Zhou, Takeshi Hara, Akitoshi Katsumata, and Hiroshi Fujita. Classification of teeth in cone-beam CT using deep convolutional neural network. *Computers in biology and medicine*, 80:24–29, 2017.

[MO11] Michael Mansfield and Colm O'sullivan. *Understanding physics*. John Wiley & Sons, 2011.

[MR08] Tanja Märkle and Manfred Rösch. Experiments on the effects of carbonization on some cultivated plant seeds. *Vegetation History and Archaeobotany*, 17(1):257–263, 2008.

[MT09] Wojciech Mokrzycki and Maciej Tatol. Perceptual difference in l* a* b* color space as the base for object colour identfication. *Image Processing & Communication Challenges*, pages 403–412, 2009.

[OFF] OFFIS. DCMTK. `https://dicom.offis.de/dcmtk.php.de`. Accessed: 2020-03-20.

[Ots79] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[Par62] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

[Pas02] G.R. Pasha. Selection of variables in multiple regression using stepwise regression. *Population*, 1(1):2, 2002.

[PD84] Thomas Porter and Tom Duff. Compositing digital images. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 253–259, 1984.

[PDM19] Huseyin Polat and Homay Danaei Mehr. Classification of pulmonary CT images by using hybrid 3D-deep convolutional neural network architecture. *Applied Sciences*, 9(5):940, 2019.

[PIV16] Giuseppe Papari, Nasiru Idowu, and Trond Varslot. Fast bilateral filtering for denoising large 3D images. *IEEE transactions on image processing*, 26(1):251–261, 2016.

[PSPM14] Jose Gustavo S Paiva, William Robson Schwartz, Helio Pedrini, and Rosane Minghim. An approach to supporting incremental visual data classification. *IEEE transactions on visualization and computer graphics*, 21(1):4–17, 2014.

[QWT] QWT. QWT. `https://qwt.sourceforge.io/`. Accessed: 2020-03-20.

[Rao48] C Radhakrishna Rao. The utilization of multiple measurements in problems of biological classification. *Journal of the Royal Statistical Society. Series B (Methodological)*, 10(2):159–203, 1948.

[RJPD14] Leszek Rutkowski, Maciej Jaworski, Lena Pietruczuk, and Piotr Duda. The CART decision tree for mining data streams. *Information Sciences*, 266:1–15, 2014.

[RLC+09] Jiangtao Ren, Sau Dan Lee, Xianlu Chen, Ben Kao, Reynold Cheng, and David Cheung. Naive Bayes classification of uncertain data. In *2009 Ninth IEEE International Conference on Data Mining*, pages 944–949. IEEE, 2009.

[RRT99] Peter J Rousseeuw, Ida Ruts, and John W Tukey. The bagplot: a bivariate boxplot. *The American Statistician*, 53(4):382–387, 1999.

[Sch13] Robert E Schapire. Explaining adaboost. In *Empirical inference*, pages 37–52. Springer, 2013.

[Sco79] David W Scott. On optimal and data-based histograms. *Biometrika*, 66(3):605–610, 1979.

[SGS95] Georgios Sakas, Marcus Grimm, and Alexandros Savopoulos. Optimized maximum intensity projection (MIP). In *Rendering Techniques' 95*, pages 51–63. Springer, 1995.

[Sha96] Linda G Shapiro. Connected component labeling and adjacency graph construction. In *Machine Intelligence and Pattern Recognition*, volume 19, pages 1–30. Elsevier, 1996.

[Sil86] Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.

[Sim19] Michael G Simpson. *Plant systematics*. Academic press, 2019.

[Sl] Scikit-learn. Scikit-learn. `https://scikit-learn.org/stable/`. Accessed: 2020-03-20.

[SL91] S. Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[Smi02] Brian Smits. Efficient bounding box intersection. *Ray tracing news*, 15(1), 2002.

[SS10] Hideaki Shimazaki and Shigeru Shinomoto. Kernel bandwidth optimization in spike rate estimation. *Journal of computational neuroscience*, 29(1-2):171–182, 2010.

[TAS08] Molka Troudi, Adel M Alimi, and Samir Saoudi. Analytical plug-in method for kernel density estimator applied to genetic neutrality study. *EURASIP Journal on Advances in Signal Processing*, 2008(1):739082, 2008.

[UC96] Graham Upton and Ian Cook. *Understanding statistics*. Oxford University Press, 1996.

[UOG+16] Mariano Ucchesu, Martino Orrù, Oscar Grillo, Gianfranco Venora, Giacomo Paglietti, Andrea Ardu, and Gianluigi Bacchetta. Predictive method for correct identification of archaeological charred grape seeds: support for advances in knowledge of grape domestication process. *PloS one*, 11(2):e0149814, 2016.

[VdV18] M. Van der Veen. Archaeobotany: The archaeology of human-plant interactions. *The science of Roman history: Biology, climate, and the future of the past*, pages 53–94, 2018.

[Was] Michael Waskom. Seaborn. `https://seaborn.pydata.org/`. Accessed: 2020-03-20.

[WBP+16] Tzu-Ching Wu, Samuel A Belteton, Jessica Pack, Daniel B Szymanski, and David M Umulis. LobeFinder: a convex hull-based method for quantitative boundary analyses of lobed plant cells. *Plant physiology*, 171(4):2331–2342, 2016.

[Węg18] Stanisław Węglarczyk. Kernel density estimation and its application. In *ITM Web of Conferences*, volume 23, page 00037. EDP Sciences, 2018.

[Wei05] Jacob L Weisdorf. From foraging to farming: explaining the Neolithic Revolution. *Journal of Economic surveys*, 19(4):561–586, 2005.

[WJ93] Matt P. Wand and M. Chris Jones. Comparison of smoothing parameterizations in bivariate kernel density estimation. *Journal of the American Statistical Association*, 88(422):520–528, 1993.

[Wu19] Ximing Wu. Robust likelihood cross-validation for kernel density estimation. *Journal of Business & Economic Statistics*, 37(4):761–770, 2019.

[Yan95] Tai-Hung Yang. *Characterization of surface profiles using discrete measurement systems*. Dissertation, Iowa State University, 1995.

[YLY+20] Kun-Hsing Yu, Tsung-Lu Michael Lee, Ming-Hsuan Yen, SC Kou, Bruce Rosen, Jung-Hsien Chiang, and Isaac S Kohane. Reproducible machine learning methods for lung cancer detection using computed

tomography images: Algorithm development and validation. *Journal of Medical Internet Research*, 22(8):e16709, 2020.

[ZF14] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[ZGWP16] Hanli Zhao, Dandan Gao, Ming Wang, and Zhigeng Pan. Real-time edge-aware weighted median filtering on the GPU. *Computers & Graphics*, 61:11–18, 2016.

[Zha04] Harry Zhang. The optimality of naive Bayes. *Proceedings of the Seventeenth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2004*, 1(2):3, 2004.

[Zho09] Zhi-Hua Zhou. Ensemble Learning. *Encyclopedia of biometrics*, 1:270–273, 2009.

[ZW16] Budi Zhao and Jianfeng Wang. 3d quantitative shape analysis on form, roundness, and compactness with μct. *Powder technology*, 291:262–275, 2016.